TITLE:

Webアプリケーションに対するサイバー攻撃の効率的な検知(Dissertation_全文)

AUTHOR(S):

鐘本, 楊

CITATION:

鐘本, 楊. Webアプリケーションに対するサイバー攻撃の効率的な検知. 京都大学, 2020, 博士(情報学)

ISSUE DATE:

2020-03-23

URL:

https://doi.org/10.14989/doctor.k22573

RIGHT:

京都大学学術情報リポジトリ
KURENAI 紅
Kyoto University Research Information Repository

Web アプリケーションに対する サイバー攻撃の効率的な検知
Efficient Detection of Cyber Attacks against Web Applications

鐘本　楊

2020 年 2 月

概要

Web アプリケーションは人々の生活に広く利用されており，社会基盤の 1 つを担っている．Web アプリケーションは一般に外部に公開されており，攻撃者も容易にアクセスが可能である．そのため，Web アプリケーションに対する攻撃は日々，大量に発生している．これらの攻撃の検知には侵入検知システム (IDS) を利用することが一般である．IDS は利用する特徴量の違いから 2 種類に大別される．ネットワーク通信に発生するリクエストやレスポンスを特徴量として検知する NIDS および Web アプリケーションが動作する OS のシステムコールや DB アクセスに行う SQL クエリ発行といったホストイベントの挙動を特徴量として検知する HIDS である．IDS は攻撃を検知するとアラートとして通知を行う．このアラート通知を受けて対応を行うのが Web サーバの管理者やセキュリティに関するアラートを専門的に扱う Security Operation Center (SOC) 分析者である．サーバ管理者や SOC 分析者は IDS のアラート通知を受けて，攻撃による被害や影響を人手で調査する．攻撃による被害を最小化するためには成功している攻撃に対してできるだけ迅速な対応がなされることが望ましい．しかし，現在の NIDS や HIDS のアラート通知にはそれぞれ，誤検知や失敗した攻撃によって大量に発生してしまうこと，あるいは，攻撃の原因や被害に関する情報がアラート関連づいていないことにより，サーバ管理者や SOC 分析者がアラート通知に対応する際の手間となっている．そのため，アラート通知を受けても迅速に対応できていないのが現在のセキュリティオペレーションにおける問題である．本研究ではこの問題を 3 つの課題に大別する．NIDS における誤検知アラートが発生する課題，HIDS におけるアラートに攻撃の情報が関連づかない課題および NIDS におけるアラートに攻撃の成否に関する情報がない課題の 3 つである．

これらの課題を鑑み，本研究では Web アプリケーションに対する攻撃をより効率的に検知する仕組みについて検討した．これら 3 つの課題に対してそれぞれの課題解決に貢献する手法を提案する．

NIDS の誤検知の課題については異常検知に基づく正確な攻撃検知手法を提案する．HTTP リクエストの各要素の文字列そのものではなく文字列の構造に着目して学習を行うことで，誤検知を低減する．評価の結果から既存の異常検知手法に比べ高精度で処理速度も実用的な手法であることを示す．

HIDS におけるアラートに攻撃の原因である HTTP リクエストが関連づかないという課題については HIDS アラートと HTTP リクエストの関連付け手法を提案する．HIDS の入力であるシステムコールや SQL クエリ発行などのホストイベントをそれらを発生させた HTTP リクエストを処理したスレッドの ID と高精度な処理開始時刻および処理終了時刻に基づいて関連付けを行うことで，HIDS で検知した際に，管理者が攻撃対象の Web アプリケーションの URL や攻撃元の IP アドレスを特定できるようにする．評価では，提案手法が誤関連付けをすることがなく，Web アプリケーションに与えるパフォーマンス低下を 5% 程度に抑えた実用的な手法であることを示す．

NIDS におけるアラートに攻撃の成否に関する情報がないという課題については攻撃コードのエミュレーションを行い，攻撃成功時の痕跡を抽出し，この痕跡が HTTP レスポンスに含まれるか否かで攻撃の成否を判定する手法を提案する．提案手法の精度・性能評価結果および発見した攻撃事例から提案手法の実用性を示す．

その結果，Web アプリケーションに対する攻撃においてサーバ管理者や SOC 分析者の対応が不要となる攻撃によるアラート通知を減少させたり，アラートの対処に必要な情報を関連づけることに貢献した．今後も Web アプリケーションに対するサイバー攻撃が増加し続けることが予想されるが，本研究で提案する手法を IDS に取り入れることで，IDS のアラートに対応するサーバ管理者や SOC 分析者の手間や対応コストの低減につながることが期待できる．

Abstract

Web applications are essential for people's daily life and they are play an important role as social infrastructure. Because web applications are publicly accessible, attackers can easily reach to them and find its vulnerability. As a result, attacks against web applications occur frequently. Intrusion Detection System (IDS) is used to detect attacks against web applications. IDS can be roughly classified into two types based on the difference in the features they used. Network-based IDS (NIDS) leverages requests and responses that occur in network communications. Host-based IDS (HIDS) leverages host events such as system calls or SQL queries. IDS notify users of alerts when an attack is detected. Server administrators and Security Operation Center (SOC) analysts receive IDS alerts and investigate the damage and effects of attacks manually. To minimize the damage of an attack, it is required that a successful attack be treated as quickly as possible. However, current NIDS and HIDS have problems with alert notifications: NIDS alerts are generated in large quantities and HIDS alerts do not correlated with HTTP request information. As a result, it becomes difficult for server administrators and SOC analysts to investigate all alerts in a limited amount of time. These problems can be divided into three issues: issue of false positive detection of NIDS, issue where HTTP request information is not correlated to alerts in HIDS and issue in which there is no information on the success or failure of the attack in the alert in NIDS.

To cope with these issues, this thesis proposes a methodology to more efficiently handle attacks against web applications. We propose a method that contributes to solving these three issues.

For the NIDS false alarm issue, we propose a more accurate attack detection algorithm based on anomaly detection. We focus on the character structure of each element of the HTTP request to create more precise models. The evaluation results show that the proposed method has higher accuracy than the existing methods, and that the performance is practical.

Regarding the issue that attack information is not related to alerts in HIDS, we propose a method for correlating HIDS alerts with HTTP requests. By correlating system calls and SQL query events with the ID of the thread that processed the HTTP request and the accurate timestamp of processing start and processing end, proposed method enables the administrator to specify the URL of the target web application and the IP address of the attack source when an attack detected by HIDS. The evaluation shows that the proposed method does not cause incorrect correlation and the performance degradation to web applications is lower than 5%.

Regarding the issue that the alert of NIDS has no information about the result of the attack, we propose a method to extract indicator of compromise which indicate the attack is successful by emulating attack code. The proposed method determines the success or failure of an attack based on whether the indicator is observed in the HTTP response or not. The accuracy and performance evaluation results show the proposed method is practical.

As a result, by reducing alert notifications or lowering the priority of handle to attacks that do not require to investigate, the tasks of server administrators and SOC analysts is reduced, leading to more efficient incident response.

目次

図目次

表目次

第 1 章

緒論

1.1 研究背景

インターネットの発展を通じて，Web アプリケーションは預金や決済をはじめとする重要なサービスに利用されるほど社会基盤の 1 つを担っている．Web アプリケーションは一般に外部に公開されており，誰でもアクセスできるため，攻撃者も容易にアクセスが可能である．そのため，Web アプリケーションに対する攻撃は日々，大量に発生している．Web アプリケーションに対する攻撃の検知には侵入検知システム (IDS) というセキュリティ装置を利用することが一般である．IDS は利用する特徴量の違いからネットワーク型 IDS (NIDS) とホスト型 IDS (HIDS) の 2 種類に大別される．NIDS は，ネットワーク通信に発生するリクエストやレスポンスを特徴量として検知を行う．HIDS は，Web アプリケーションが動作するオペレーティングシステム (OS) のシステムコールやデータベース (DB) にアクセスする際に利用する SQL クエリ発行などのサーバ内で発生するイベント (ホストイベント) を特徴量として検知を行う．NIDS/HIDS は攻撃を検知するとアラートとして通知を行う．このアラート通知を受けて攻撃に対応するのが，Web サーバの管理者やセキュリティに関するアラートを専門的に扱う SOC (Security Operation Center) 分析者である．サーバ管理者や SOC 分析者は，NIDS/HIDS のアラート通知を受けて，攻撃による被害や影響を人手で調査する．攻撃による被害を最小化するためには成功している攻撃に対してできるだけ迅速な対応がなされることが望ましい．迅速な対応のため，本研究では IDS のアラート通知には以下の 2 つの要件が必要だと考える．

要件 1　アラートが正確に攻撃を示しており，かつその攻撃の成否に関する情報が存在する
要件 2　アラートに攻撃の原因と被害に関する情報の両方が存在する

要件 1 の理由は，本研究では人手で優先的に対応することが必要な攻撃は成功した攻撃であるという考えにある．図 1.1にその概念を示す．ネットワーク通信の大半は正常な通信

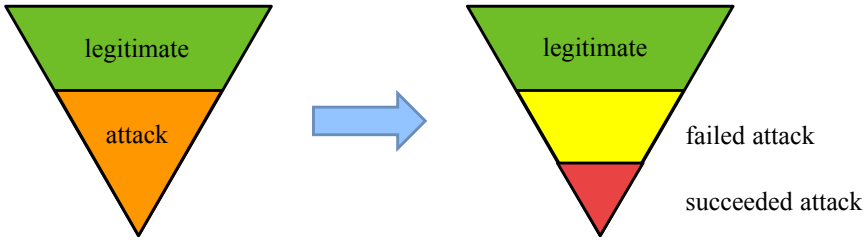

図1.1: 要件 1 における概念

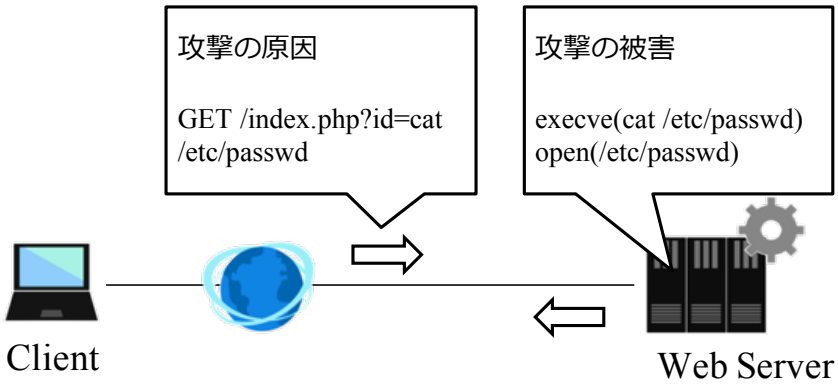

図1.2: 要件 2 における概念

(legitimate) であり，一部分が攻撃 (attack) であり IDS が検知すべき対象である．本研究では更に攻撃をその結果から成功した攻撃 (succeeded attack) と失敗した攻撃 (failed attack) の 2 つに大別する．人手で最優先に対応すべきものは成功した攻撃であるため，アラートに攻撃の成否に関する情報があれば，成功した攻撃に優先的に対応し，失敗した攻撃には対応する優先度を下げることが可能となり，被害の最小化につながると考える．

要件 2 の理由は，攻撃の原因と被害に関する情報は攻撃の迅速な暫定対処につながるという考えにある．図 1.2にその概念を示す．攻撃が成功して被害が発生した際はサーバ管理者や SOC 分析者は被害が再発しないよう暫定対処を行う．暫定対処とは例えば，攻撃の標的となった URL に対するアクセスを禁止したり，攻撃の送信元となっている IP アドレスとの通信を遮断したりなど，根本的な解決ではないが一時的に同じ攻撃によって再度被害が発生しないようにするための処置である．そのため，攻撃の原因である標的アプリケーションの URL や送信元 IP アドレスなどの情報を含む HTTP リクエストの情報が必要である．

しかし，現在の NIDS や HIDS のアラート通知は本研究で述べる要件に対して満たしておらず，サーバ管理者や SOC 分析者の手間を発生させてしまっている．より具体的には以下に示す 3 つの課題が存在する．

課題 1　NIDS における誤検知アラートが発生する

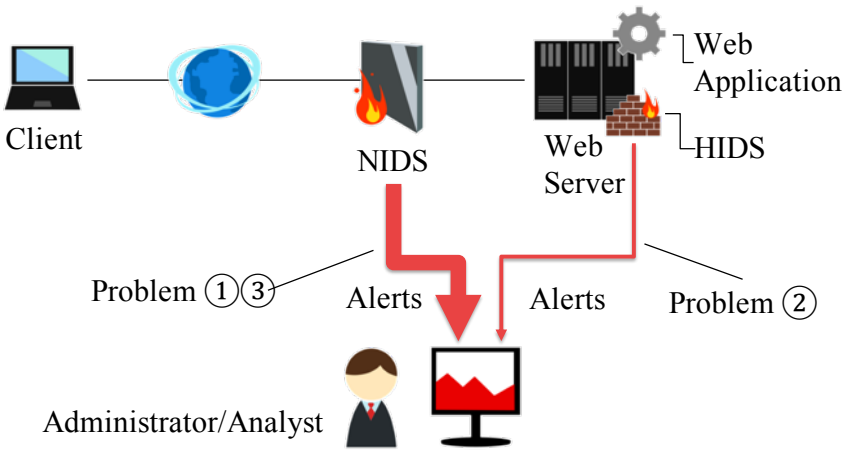

図1.3: Web アプリケーションにおける攻撃検知とその課題

課題 2　HIDS におけるアラートに攻撃の情報が関連付かない
課題 3　NIDS におけるアラートに攻撃の成否に関する情報がない

課題 1 および**課題 3** は**要件 1** に対する課題であり，**課題 2** は**要件 2** に対するものである．図 1.3に本研究が対象とする Web アプリケーションにおける攻撃検知とその際の課題の概念図を示す．

NIDS において最も検知の効率を低下させてしまうのが正常の通信を攻撃として誤検知してしまう課題である (**課題 1**)．既存の検知手法 [1, 2, 3, 4, 5, 6, 7] ではいずれも HTTP リクエストに出現する文字列そのものを特徴として用いるため，学習モデルを作成する際に与えられた学習データが網羅的でないと，正常な入力値を攻撃として検知してしまう誤検知が多く発生する．誤検知が大量に発生してしまうと，アラート通知を受けたサーバ管理者や SOC 分析者は正常な通信に対しても被害の有無の確認をしなければならず調査に手間がかかってしまう．

誤検知が発生抑制されれば，サーバ管理者や SOC 分析者は真に攻撃に関するアラートのみに集中することができる．しかしながら，攻撃が大量に発生する現在の状況では，攻撃に関するアラート全てを人手で対応することは困難である．本来，人手で対応すべきものは攻撃が成功した場合のみである．そのため，攻撃の成否が判別できればサーバ管理者や SOC 分析者は成功した攻撃のみに対応したり，成功と判定した攻撃に優先的に対応したりすることが可能となり，被害を引き起こす攻撃に対してより迅速に対応することが可能となるため，被害の最小化につながる．

攻撃の成否を判別するには攻撃の意図に応じた攻撃の痕跡を捉えることが必要である．本研究では Web アプリケーションに対する攻撃を 4 つの意図に大別して考える．脆弱性確認，情報漏洩，乗っ取り，改ざんである．これらの意図についての詳細は 2.6節にて解説する．それぞれの攻撃の意図に対する痕跡の出現する場所の定義を表 1.1に示す．脆弱性確認を行う攻撃

表1.1: 攻撃の意図毎の痕跡の出現場所

攻撃の意図	痕跡の出現場所	
	ネットワーク通信	サーバ挙動
脆弱性確認	✓	✗
情報漏洩	✓	✓
乗っ取り	✓	✓
改ざん	✗	✓

は攻撃者がインターネット上に存在する大量のサーバから効率良く攻撃可能なサーバを探索することが狙いであり，サーバに被害を発生させることが狙いではないため，サーバの挙動に痕跡は出現せず，通信に痕跡が残ると考える．一方，改ざんを行う攻撃は HTTP レスポンス等の通信に攻撃の痕跡が出現することが少ないため，成否を判断するためにはサーバ内の挙動を観測して痕跡を見つけることが必要になる．つまり，成否を判定する方法は，痕跡の出現する場所によって異なるアプローチが必要にある．

改ざんを含むサーバの挙動に痕跡が現れる攻撃の成否を判定するには，HIDS を活用することができる．HIDS はホスト内の情報を活用して攻撃検知を行うため，HIDS が検知するということは攻撃が成功してサーバに被害が発生しているとして捉えることができる．しかし，HIDS のアラート通知には攻撃の原因となる HTTP リクエストに関する情報が関連付いていない．つまり，HIDS のアラート通知にはどのようなファイルが改ざんされたかあるいはどのようなコマンドが不正に実行されたかといった被害については捉えることが可能だが，その被害がどのような攻撃 HTTP リクエストによって引き起こされたのかといった原因については捉えることができない．そのため，サーバ管理者や SOC 分析者は攻撃が再発しないよう対応する際に，人手で被害を引き起こした原因となった HTTP リクエストを調査する手間が発生してしまう (**課題 2**) .

HIDS はその性質上，サーバに対する改変や導入によるサーバのパフォーマンス低下は免れないため，これらの制約は導入や運用する際の障壁となる．また，脆弱性確認を行う攻撃はネットワーク通信のみに痕跡が現れるため攻撃の成否を判定できない．そのため，NIDS と同様にネットワーク通信を特徴量として攻撃の成否が判定できることが望ましい．しかし，調査した結果，既存の NIDS の手法は攻撃の成否に着目しているものは存在しない (**課題 3**) .

これらの背景のもと，本研究では，NIDS の誤検知を抑える手法，HIDS のアラートに対して HTTP リクエスト情報を関連付ける手法および，ネットワーク通信から攻撃の成否を判定する手法を検討することで，Web アプリケーションに対する攻撃のより効率的な検知を目標とする．

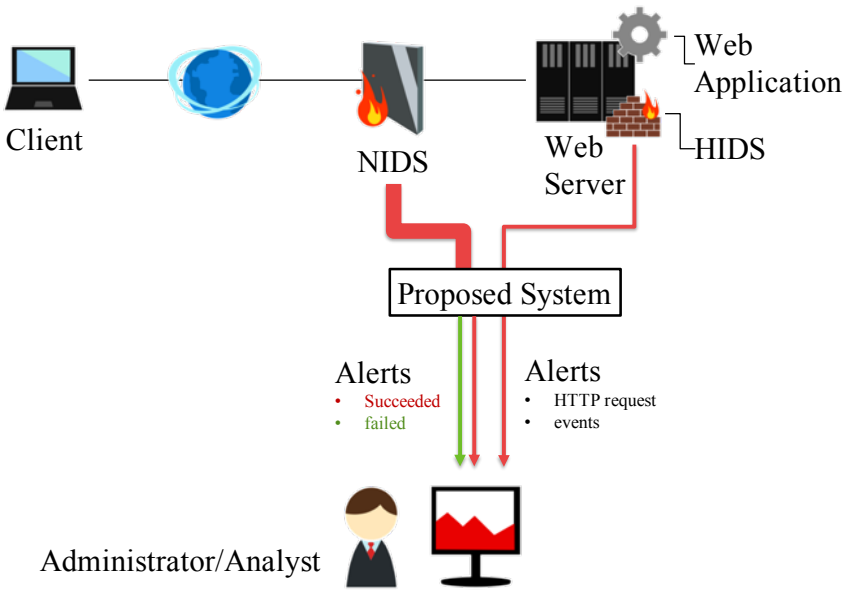

図1.4: 本研究が目指す世界像

1.2　研究の位置づけと研究方針

本研究では，これらの課題を鑑み，Web アプリケーションに対する攻撃をより効率的に検知する仕組みについて検討した．図 1.4に本研究で目指す世界像を示す．NIDS および HIDS のアラート通知が本研究で提案する手法が実装されたシステムを介することで，誤検知に対するアラートが除外されたり，アラートに不足している情報が関連付けられたり，成功した攻撃に関するアラートと失敗した攻撃に関するアラートが区別できるようになる．

課題 1 の NIDS の誤検知の課題については，異常検知に基づくより正確な検知手法を提案する．HTTP リクエストの各要素である URL パラメタ，HTTP ヘッダの値そのものではなく，その値の構造を学習することで，誤検知を低減する．評価では，既存の異常検知手法に比べ提案手法の方が精度が高く，また性能も実用的な処理速度であることを示す．

課題 2 の HIDS におけるアラートに攻撃の情報が関連付かないという課題については HIDS アラートと HTTP リクエストの関連付け手法を提案する．HIDS の入力であるシステムコールや SQL クエリ発行などのイベントを，それらを発生させた HTTP リクエストを処理したスレッドの ID と高精度な処理開始および終了時刻に基づいて関連付けを行うことで，HIDS で検知した際に管理者が攻撃対象の Web アプリケーションの URL や攻撃元の IP アドレスを特定できるようにする．評価では，提案手法が誤った関連付けをすることがなく，Web アプリケーションに与えるパフォーマンス低下を 5% 程度に抑えた実用的な手法であることを示す．

課題 3 の NIDS におけるアラートに攻撃の成否に関する情報がないという課題については攻撃コードのエミュレーションを行い，攻撃成功時の痕跡を抽出し，この痕跡が HTTP レスポンスに含まれるか否かで攻撃の成否を判定する手法を提案する．提案手法の精度・性能評価結果，および発見した攻撃事例から提案手法の実用性を示す．

その結果，Web アプリケーションに対する攻撃において対応が不要となる攻撃によるアラート通知を減少させるあるいは対応優先順位を下げること，さらにアラートに攻撃の原因と被害の両方に関する情報を付与することを実現した．今後も Web アプリケーションに対するサイバー攻撃が増加し続けることが予想されるが，本研究で提案する手法を IDS に取り入れることで，管理者や SOC 分析者の対応コストの低減につながることが期待できる．

1.3　全体構成

本論文の構成を述べる．本論文は 7 章で構成され，各章の内容は以下の通りである．1章では緒論として，本研究を行うに至った背景と目的および本研究の位置づけと研究方針，そして全体の構成について説明する．2章では，本研究における基本概念と従来研究について述べる．基本概念は，本研究の前提となる Web アプリケーションやその脆弱性，脆弱性に対する攻撃，攻撃検知の仕組みおよび，検知した攻撃に対するセキュリティオペレーションの概念について述べる．3章では，**課題 1** の NIDS の誤検知の課題に対して，異常検知に基づくより正確な攻撃検知アルゴリズムについて述べる．4章では，**課題 2** の HIDS におけるアラートに攻撃の原因となる HTTP リクエストが関連付かないという課題に対して，HIDS アラートと HTTP リクエストの関連付け手法を提案する．5章では，**課題 3** の NIDS におけるアラートに攻撃の成否に関する情報がないという課題に対して，攻撃コードのエミュレーションを行い，攻撃成功時の痕跡を抽出し，この痕跡が HTTP レスポンスに含まれるか否かで攻撃の成否を判定する手法について述べる．6章では，5章の手法を拡張し，未対応であるリモート型シェルコードによる攻撃の成否を判定する手法を提案する．7章では，3章から 6章の各章の実験で得られた結論と，Web アプリケーションに対する攻撃の検知に関する今後の展望について述べる．

第 2 章

基本概念と従来研究

2.1 緒言

本章では，本研究の前提となる Web アプリケーションやその脆弱性，脆弱性に対する攻撃，攻撃の意図，攻撃検知の仕組みおよび，検知した攻撃に対するセキュリティオペレーションの概念について述べる．

2.2 Web アプリケーション

Web の始まりは 1990 年であり [8]，当初は Web ページという静的なコンテンツに掲載された情報を照会することが主な利用用途だった．現在では静的なコンテンツの照会のみではなく，ユーザが情報を提供したり，更新したりする Web アプリケーションが普及している．インターネット空間にて Web アプリケーションを介してサービスを行うのが Web サイトである．Web サイト数は継続的に拡大しており，Web サイト数推移や Web サーバの市場シェアを観測している Netcraft によると Web サイト数は 1995 年から 2019 年で約 64 万倍に増加し，現在では約 12 億もの Web サイトが存在する [9].

Web アプリケーションは，ユーザからの要求を受け取り，ファイルや DB の内容を取得して応答したり，更新したりする処理を行う．例えば，ブログを開設するための Web アプリケーションとして WordPress [10] や Joomla [11] などが存在する．一方，Web サーバは，ユーザからの要求を Web アプリケーションに転送する役割を果たしている．例えば，Apache HTTP Server [12] や nginx [13] などが存在する．図 2.1に一般的な Web アプリケーションと Web サーバの構成を示す．企業や組織など大規模な環境では，(a) の構成のように高負荷にも耐えられるように Web サーバと Web アプリケーションサーバを別々のサーバで構成する．個人などの小規模な環境ではコンピュータ資源が限られているため，(b) の構成のように Web アプリケーションが Web サーバ内で動作する場合も多い．本研究では上記で述べた Web アプリ

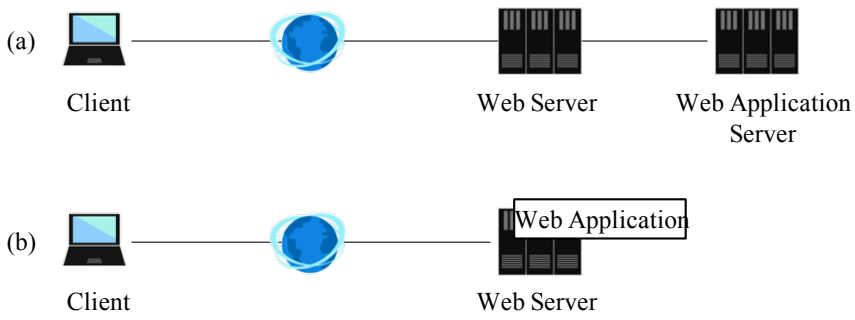

図2.1: Web アプリケーションと Web サーバの関係

ケーションに対する攻撃を対象とする．

2.3　HyperText Transfer Protocol

HTTP (HyperText Transfer Protocol) は，クライアントが Web サーバと通信する際に使用する通信プロトコルである．Web の始まりと共に設計され，利用が広まり，現在では Web アプリケーションは全て HTTP を利用した通信を行っている．1991 年，HTTP の仕様として最初公開されたバージョンが HTTP/0.9 である [14]．その後マイナーバージョンの更新を繰り返し，現在では HTTP/1.1 が最も広く利用されている．その後継となる HTTP/2 [15] は 2015 年に仕様が定まり公開され，徐々に移行が進んでいる状況である．HTTP/1.1 に比べてフレームとストリームによる通信の多重化，ヘッダの圧縮およびサーバプッシュといった高速に通信するための技術が取り入れられており，今後の普及が期待される．

HTTP を利用する通信はリクエストとレスポンスから成り立っている．リクエストおよびレスポンスは更に複数の構成要素に分解することが可能である．HTTP リクエストは更に，メソッド，URL，バージョン，ヘッダ，ボディに分解でき，HTTP レスポンスは更に，ステータス，ヘッダ，ボディに分解される．Web アプリケーションは HTTP を介してリクエストおよびレスポンスを送受信することでユーザから要求された処理を行っている．

2.4　Web アプリケーションの脆弱性

攻撃には既知攻撃もあれば，未知攻撃の可能性も存在する．既知攻撃とは過去に観測されたことのある攻撃であり，未知攻撃とは未発見の脆弱性への攻撃のことである．表 2.1は脆弱性情報データベースである CVE (Common Vulnerabilities and Exposures) [16] から収集した過去 10 年間に毎年発見された脆弱性の件数とそのうち Web アプリケーションに関する脆弱性の件数である．セキュアコーディング [17] やソースコード分析 [18, 19, 20] などの技術が発

表2.1: 過去 10 年間に発見された脆弱性の件数

年	全て	Web 関連
2009	5,297	1,437
2010	4,639	1,160
2011	4,150	1,104
2012	5,289	1,582
2013	5,186	1,505
2014	7,937	2,148
2015	6,487	1,784
2016	6,447	1,228
2017	14,649	2,308
2018	15,998	2,133
2019	17,314	2,183

展してきていても，毎年発見される Web アプリケーションに関する脆弱性の件数は依然として減少していないのが現状である．

2.5 Web アプリケーションに対する攻撃

外部に公開しているサーバは一般の利用者だけでなく，攻撃者からアクセスされやすく攻撃にさらされやすい．2.4節で述べたように，Web アプリケーションの脆弱性は常に発見されている．そのため，攻撃者も脆弱な Web アプリケーションが公開されていないか日々調査を行っている．攻撃者は効率良く Web アプリケーションの脆弱性を発見するために，ボットを活用したスキャン [21] や Google Dorking [22] を行なっているため，Web 空間では攻撃が日々大量に発生している．セキュリティ企業である Positive Technologies のレポートによると，IT 企業，銀行，政府の Web サイトに対して 1 日約 1000 件にも上る攻撃が行われている [23]．また，世界でも有数の大規模 CDN (Contents Delivery Network) 事業者である Akamai の観測によると，2015 年には 13 億件だった攻撃数が，2017 年には 35 億件へと増加しており，近年では 1 日あたり 1,000 万件以上もの攻撃が発生している [24]．

Web アプリケーションに対する攻撃は，実装上の脆弱性を悪用する攻撃，仕様上の脆弱性を悪用する攻撃とサービス不能を狙う攻撃の 3 種類に大別できる．以下，各種類の攻撃について詳説する．

2.5.1　実装上の脆弱性を悪用する攻撃

実装上の脆弱性を悪用する攻撃とは Web アプリケーションに潜むプログラムコード上のミスによって引き起こされる脆弱性に対する攻撃である．例えば，SQL インジェクション，OS コマンドインジェクションや XSS (Cross-Site Scripting) などが挙げられる．

SQL インジェクションとは，Web アプリケーションが想定しない SQL 文を DB に実行させ，DB を改ざんしたり，情報を奪取したりする攻撃である．例えば，Web アプリケーションが以下の SQL 文を実行するとする．

```
SELECT name FROM users WHERE id = '$_GET["id"]';
```

この SQL 文はユーザが指定した id に該当するユーザの名前のみを取得するものであり，$_GET["id"]がユーザが入力する値である．通常，id には数値が入力されることが想定されるが，' OR 1=1 --'のように SQL 文を入力値とすることで以下のような SQL 文が実行される．

```
SELECT name FROM users WHERE id = '' OR 1=1 --'';
```

この SQL 文では 1=1 の部分が常に真となるから，DB に存在する全てのユーザの名前が取得され情報漏えいにつながる可能性がある．

OS コマンドインジェクションは，Web アプリケーションが想定しない OS コマンドを OS に実行させ，ファイルを改ざんしたり，設定を変更したりする攻撃である．例えば，サーバに対する死活監視のため以下のように ping コマンドを実行する Web アプリケーションが存在するとする．

```
ping -c 4 $_GET["host"];
```

$_GET["host"]には IP アドレスや FQDN のようなホスト名の入力が期待されるが，; cat /etc/passwdのようにコマンドを入力することで以下のようなコマンドが実行される．

```
ping -c 4 ; cat /etc/passwd;
```

このコマンドでは ping -c 4 の部分は引数不足で実行できず，その後に続く cat /etc/passwd の部分が実行され，意図しない情報が出力されてしまう．

XSS とは，Web アプリケーションの出力が攻撃者によって操作できる際に発生する攻撃である．SQL インジェクションや OS コマンドインジェクションと異なり，脆弱性が存在するのはサーバ側であるが，被害を受けるのはクライアント側である．例えば以下のようなメッセージを表示する Web アプリケーションがあるとする．

```
<?php echo '<html>Welcome! $_GET["name"]</html>' ?>;
```

`$_GET["name"]`に Alice や Bob のような名前が入力されることが想定されるが，`<script>alert("Hacked")</script>`といった HTML や JavaScript を入力することで閲覧者の意図に反してポップアップを表示させることが可能である．

2.5.2 仕様上の脆弱性を悪用する攻撃

仕様上の脆弱性を悪用する攻撃とは Web アプリケーションの仕様上のミスによって引き起こされる脆弱性に対する攻撃である．例えば，CSRF (Cross-Site Request Forgery) やセッションハイジャックなどが挙げられる．CSRF はユーザが意図しないアクセスを攻撃者が行わせる攻撃である．例えば，ユーザのログイン後に，以下のような HTTP リクエストによって送金を行う Web アプリケーションを想定する．

```
GET /transfer.php?to=bob&amount=10000 HTTP/1.1
```

攻撃者は以下のようなリンクをばらまくことで，そのリンクをクリックした人がログイン状態の場合，強制的に送金を行わせることができる．

```
<a href="http://bank/transfer.php?to=attacker&amount=10000">
Click here !
</a>
```

2.5.3 サービス不能を狙う攻撃

サービス不能を狙う攻撃とは大量のリクエストを行うことで Web アプリケーションが動作している Web サーバのリソースを消費し，サービスを不可にする種類の攻撃である．有名なものとしては，例えば，インターネット上に分散した複数のクライアントから大量にリクエストを行う DDoS (Distributed Denial of Service) 攻撃 [25] や，長期間低速にリクエストを行う SlowDoS 攻撃 [26] などが挙げられる．

2.5.4 本研究で注力する攻撃

上記のように，Web アプリケーションに対する攻撃は，その脆弱性の種類によって攻撃方法が異なる．国際的な Web アプリケーションのセキュリティコミュニティである OWASP (Open Web Application Security Project) が公開している対策すべき重要な Web アプリケーションの脆弱性 Top 10 では，SQL インジェクションや OS コマンドインジェクションのような実装上の脆弱性を悪用する攻撃を最も重視しており，喫緊の問題である [27]．そのため，本研究においても，Web アプリケーションの実装上の脆弱性を悪用する攻撃について注力する．

2.6　Web アプリケーションに対する攻撃の意図

Web アプリケーションに対する攻撃方法は様々であるが，その意図は限定的である．本研究では，Web アプリケーションに対する攻撃の意図を脆弱性確認，情報漏えい，乗っ取り，改ざんの 4 つに大別する．以下，各攻撃の意図について例を挙げながら説明する．

脆弱性の確認

Web アプリケーションや Web サーバに対して明確な被害を与えるのではなく，悪用できる脆弱性があることを確認する攻撃である．攻撃者はインターネット空間上にある大量の Web サーバに対する攻撃を効率化するため，まずは大量の Web サーバに対して脆弱性の確認を行なった後，脆弱性があることが確認された Web サーバに対して，情報漏えいや改ざんなどを引き起こす攻撃を行う [21]．具体的な確認方法としては，例えば，`echo vulnerable` といったコマンドを実行して，レスポンスに `vulnerable` という文字列があられるか否かで脆弱性の有無を確認する．攻撃が成功しても被害が発生するとは限らない．

情報漏えい

Web アプリケーションにおいて公開していないファイルやデータを閲覧する攻撃である．例えば，`cat /etc/passwd` コマンドを実行して，サーバに存在するユーザの一覧を取得する．

乗っ取り

今後の悪用のため，Web アプリケーションや Web サーバに制御用のプログラムを送り込み，実行する攻撃である．例えば，`wget http://x.x.x.x/bot.sh && sh bot.sh` コマンドによって，サーバを制御できるようにするプログラム `bot.sh` を外部の Web サイト `x.x.x.x` からダウンロードした後に実行する．

改ざん

サーバの設定やファイルを改変する攻撃である．例えば，`echo Hacked!! > /var/www/html/index.php` コマンドを実行して，Web アプリケーションのトップページを変更したり，あるいは `iptables -F` コマンドを実行してネットワーク接続制限を全て無効にしたりする．

2.7 侵入検知システム

Web アプリケーションに対する攻撃の検知には IDS というセキュリティ装置を利用することが一般である．IDS は攻撃を検知した場合，アラートとして通知を行う．本研究では IDS を利用する特徴量の違いから NIDS と HIDS の 2 種類に大別する．NIDS と HIDS の区別については IDS が導入される形態によって区別する概念も存在する．つまり，ネットワーク経路上に設置される IDS は NIDS であり，サーバ内に設置される IDS は HIDS であるという区別の仕方である．この場合，ネットワーク通信を特徴量として攻撃検知を行ってもそれが実施されるのがサーバ上であれば HIDS と区別される．本研究ではこの様に IDS の導入形態でなく，IDS が扱う特徴量の違いによって区別する．

2.7.1 NIDS

ネットワーク通信に発生するリクエストやレスポンスを特徴量として攻撃を検知する仕組みである．NIDS の中でも Web アプリケーションに対する攻撃検知に特化したものを Web Application Firewall (WAF) と呼ぶ．既存手法にはネットワーク通信に現れるバイト列の頻度あるいは，通信の特徴を数値化した後に異常検知を適用する手法が存在する [28, 29, 30]．これらの手法はその特徴量の取り方から通信プロトコルに依存しないで攻撃検知が可能なため，幅広いプロトコルに対する攻撃を検知することができる．Web アプリケーションの HTTP 通信に特化した異常検知手法には文字列長や n-gram 等を利用する手法 [1, 3, 4, 31, 5, 32] などが存在する．NIDS はサーバ改変のが不要であり，処理パフォーマンス低下が少ないことから導入が容易である．しかし，誤検知や失敗した攻撃によるアラート通知が大量に発生してしまうという課題があり，アラートの対応を行うサーバ管理者や SOC 分析者が調査すべきアラートが大量になってしまい，分析を行う負担が大きい．

2.7.2 HIDS

Web アプリケーションが動作する Web サーバのシステムコールや SQL クエリ発行などのホストイベントを特徴量として検知する仕組みである．そのため，攻撃によって被害が発生している事象を検知する仕組みであると考えることができる．既存手法にはシステムコールの順序やシステムコールの引数を特徴量として異常検知を行う手法 [33, 34, 35] や，Web アプリケーションに特化し，DB アクセスの際に利用される SQL クエリを特徴量として異常検知を行う手法などが存在する [36, 37]．HIDS の実用上の欠点としては，導入にはサーバ改変を要し，サーバの処理性能を低下させることにある．そのため，HIDS 導入には Web アプリケーションや Web サーバの動作に悪影響がないか検証しながら行う必要がある．

2.7.3　攻撃を遮断する IPS

Intrusion Prevention System (IPS) は攻撃を検知するのみでなく遮断も行う仕組みである．全ての攻撃を正しく検知することができれば遮断してしまうことが実現するが，攻撃を誤検知する可能性があるということは，ユーザの正常アクセスを誤遮断してしまい，サービス品質の低下につながるリスクも存在する．そのため，サービス提供者としては遮断せず，検知のみを行う IDS を利用していたり，遮断する場合においても，確実に攻撃と断定できる場合のみに限られていたりするため，IDS のアラート通知を調査する必要がある．

2.8　セキュリティオペレーション

米国 NIST では，サイバー攻撃の脅威に対するより良い対策として，NIST Cyber Security Framework (CSF) というフレームワークを定義している [38]．このフレームワークでは，対策は識別，防御，検知，対応，復旧の 5 つの段階に分かれる．識別，防御は攻撃が起きる前に行う事前対策であり，脆弱性がないようにしたり，アクセス制御により，攻撃されたとしても被害が最小限になるよう予防することを指す．しかし，事前対策は完璧ではないため，攻撃された際の事後対処も想定しておく必要がある．検知，対応，復旧は攻撃が起きたあとに行う事後対策であり，ネットワーク通信やログから攻撃の兆候を検知した場合，封じ込めの対応を行い，攻撃される前の状態に戻すことを指している．

Web アプリケーションに対して攻撃が発生した場合も同様に対応を行う必要がある．検知の役割を担う IDS が発したアラートを調査するのがサーバ管理者や SOC 分析者である．SOC とは IDS のアラートやその他のセキュリティインシデントを専門的に扱う組織であり，24 時間 365 日アラート通知を監視し，攻撃による被害がないか，被害がある場合どの程度の被害なのかを調査する．

しかし，攻撃が大量に発生する現状では，大量のアラートを受け取ることとなってしまう．アラートを確認し，インシデントが発生したかを確認する部分は人手で行うことが一般的であるため，大量のアラートの確認には多くの時間を要する．そのため，限られた人員で全てのアラートに対処することが困難になってきている．Cisco のレポート [39] によれば，組織で発見された 44% ものアラートは対処できずに見過ごされていると報告されている．しかし，多くの場合において，誤検知であるアラートや失敗した攻撃もアラートに含まれるため，大量のアラートのうち，実際に被害 (改ざん，乗っ取り，漏えい等) に至るものは，ごく一部である．そのため，Web アプリケーションに対する攻撃のセキュリティオペレーションにおいては，まずセキュリティ侵害が発生したことを示すアラートであるか否かを見極め，優先して対処できるかが課題である．

第 3 章

異常検知を活用した Web アプリケーションに対する攻撃の検知

3.1 緒言

Web アプリケーションは一般に外部に公開されており，攻撃者も容易にアクセスが可能であるため，Web アプリケーションに対する脆弱性を悪用する攻撃は日々，大量発生している．

攻撃に対する検知方法にはシグネチャ検知と異常検知の 2 つの方式が存在する．シグネチャ検知は既知の攻撃のバイト列から特徴的な部分を正規表現で表現したシグネチャをあらかじめ用意し，検知対象である HTTP リクエストがシグネチャの正規表現にマッチした場合，攻撃として検知する方式である．しかし，この方式では難読化された攻撃コードの検知や，あらかじめシグネチャを用意しておくことができない未知攻撃の検知には期待できないことが課題である．そのため，難読化された攻撃コードや未知攻撃を捉えるためには異常検知が必要である．異常検知は正常な HTTP リクエストを学習して正常モデルであるプロファイルを作成しておき，検知対象である HTTP リクエストがプロファイルと乖離した場合，攻撃として検知する方式である．そのため，たとえ攻撃コードが既知のものと異なったとしても検知することが可能であり，未知の攻撃も検出できる可能性がある．

既存の異常検知手法ではいずれも HTTP リクエストに出現する文字列そのものを特徴として用いるため，プロファイルを作成する際に与えられた学習データが網羅的でないと，正常な入力値を攻撃として検知してしまう誤検知が多く発生する．

本研究では既存手法と異なり，全ての文字列を対象として正規化を施すことで，より網羅的なプロファイルを作成し，誤検知の低下を目指す．具体的には文字列の型構造を表現した文字クラス列および文字クラス集合を特徴として導入する．実験により提案手法は既存の手法に対して検知率を同程度に維持しながら，誤検知率を減少できることを示す．

本研究の貢献は以下の通りである．

- 文字列をより抽象的に表現する方法を考案し，その方法用いた異常検知手法を提案した．
- 約 134 万件の実データに対して評価を行い，提案手法が既存手法より高精度であることを示した．
- HTTP リクエスト 1 件あたり 1 ミリ秒以内で検知処理可能であることを示し，実用性が高いことを示した．

3.2　既存手法とその課題

本研究で注目する Web アプリケーションに対する異常検知では，HTTP プロトコルの各要素を解釈した上で，異常検知を適用することが一般である．具体的には HTTP リクエストの URL パス，URL パラメタ，ヘッダ，Cookie 等，HTTP リクエストを構成する要素を分解した上で学習時は各要素ごとにプロファイルを作成し，検知時は各要素ごとに異常か否かの判定を行う．

Web アプリケーションの URL パラメタを悪用する攻撃に対して異常検知を試みた手法に Kruegel らの手法 [1] や Kim らの手法 [3] が存在する．Kruegel らの手法 [1] では，URL パラメタのみを対象として，各 URL パラメタ毎にプロファイルを作成する．プロファイルは文字列長，文字分布，文字列構造などの特徴から成る．これらの特徴から正常との乖離度合いを算出して，閾値以上である場合，攻撃として検知する．Kim らの手法 [3] では Kruegel らの手法と同じ特徴を用いるが，プロファイル毎に用いる特徴を選択するところが異なる．

Web アプリケーションの URL パラメタに着目するのではなく，HTTP リクエスト全体に着目した手法に Anagram [4], Ingham らの手法 [5], Luo らの手法 [6] や Betarte らの手法 [7] が存在する．Anagram では学習時に HTTP リクエスト全体の文字列に対して n-gram を求め，保持する．検知時は同様に HTTP リクエストに対して n-gram を求め，各 n-gram の出現割合によって攻撃か否かを判定手法である．Ingham らの手法ではまず，トークンと呼ばれる HTTP リクエストに表れる特徴的な表現を定義し，HTTP リクエストに対してトークン列を作成する．例えば，`GET / HTTP/1.1` のような HTTP リクエストがあった場合，トークン列は (`GET`, `/`, `HTTP/1.1`) となる．学習時は HTTP リクエストから作成したトークン列から各トークンの遷移モデルを作成する．上記の例の場合 `GET` → `/`，`/` → `HTTP/1.1` という遷移モデルが作成される．検知時は検知対象の HTTP リクエストのトークン列と学習した遷移モデルの乖離度合いから攻撃を検知することを試みている．Luo らの手法では HTTP リクエストをバイト値に基づいて特徴ベクトル化する．次に並列的に学習器を 1 つずつ順番に構築していく XGBoost [40] を利用して学習モデルを作成し，その学習モデルとの乖離をもとに攻撃を検知する．Betarte らの手法ではあらかじめ用意した特徴的な文字列を用意し，HTTP リクエストにその文字列が含まれる数を計算し，特徴ベクトル化する．特徴ベクトルに対して，混合ガ

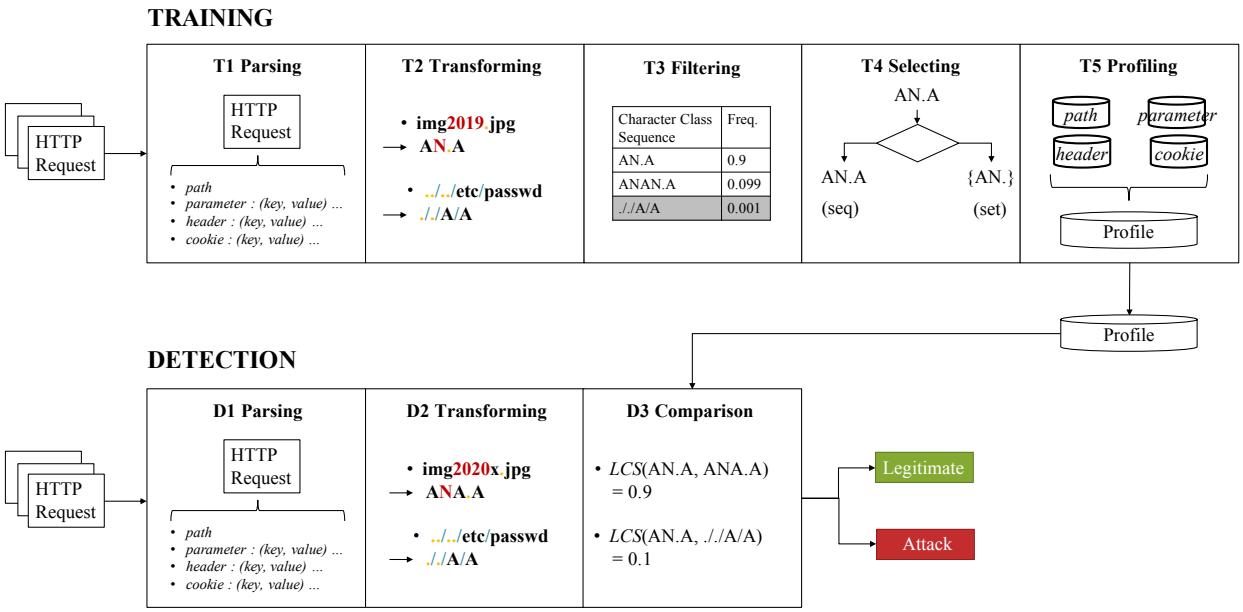

図3.1: 提案手法の処理概要

ウスモデルを利用して学習モデルを作成し，そのモデルとの乖離をもとに攻撃を検知する．

特徴の捉え方はいずれの既存手法でも異なるが，学習の際に入力値の文字列そのものを用いている点では共通している．Kruegel らの手法 [1] や Kim らの手法 [3] では文字列の構造を表す際に，文字列そのものの遷移モデルを利用している．例えば，文字列 `abc` という入力があった場合，`a` → `b`，`b` → `c` という遷移が学習される．Anagram [4] や Ingham らの手法 [5] では特徴に文字列の n-gram やトークンを用いているが，上記の手法と同様に文字列そのものが特徴となっている．また，Luo らの手法 [6] や Betarte らの手法 [7] では HTTP リクエストに現れる文字列を特徴ベクトル化して特徴量として扱っている．このように，既存手法はいずれも学習時に大量のデータを与えることが可能な場合，あるいは Web アプリケーション自体が静的コンテンツのみで構成されるような単純な仕組みの場合には適用できるが，動的な Web アプリケーションにおいては誤検知が増加してしまうという課題が存在する [41]．

3.3　提案手法

既存手法の課題は HTTP リクエストの各要素の入力値に現れる文字列そのものを用いてプロファイルを作成することに由来していた．提案手法では入力された文字列全てを正規化することでこの問題を解消することを目指す．具体的には提案手法では入力値の文字列を文字種類ごとに定義した文字クラスへの変換を行い，文字列の構造を表現するプロファイルを作成する．

提案手法の概要を図 3.1に示す．提案手法は学習と検知の 2 つの処理に大別される．学習時は正常な HTTP リクエストを入力とし，HTTP リクエストを要素ごとに分解し，要素ごとに

プロファイルを作成する．検知時は学習時と同様に入力された HTTP リクエストを要素に分解し，要素ごとに学習したプロファイルと照合し，乖離度を求める．乖離度があらかじめ指定した閾値を超えていれば攻撃と判定し，そうでなければ正常と判定する．以下，図 3.1の各ステップで示す学習と検知の詳細について，各小節で述べる．

3.3.1　学習

提案手法の学習プロセスは以下の 4 つのステップで構成される．

T1: Parsing

入力された HTTP リクエストを URL パス，URL パラメタの Key と Value，ヘッダの Key と Value，Cookie の Key と Value の要素ごとに分解する．例えば，以下の HTTP リクエストがあった場合，

```
GET /index.php?id=1&name=alice HTTP/1.1
Host: example.net
User-Agent: Mozilla/5.0
Cookie: PHPSESSID=1234;
```

URL パスは`/index.php`，URL パラメタは {`id: 1`, `name: alice`} といった Key-Value 形式，ヘッダは {`Host: example.net`, `User-Agent: Mozilla/5.0`} といった Key-Value 形式，Cookie は {`PHPSESSID: 1234`} といった Key-Value 形式に分解される．

T2: Transforming

各要素の入力値の文字列を文字クラス列に変換する．文字クラスとは文字列の型を意味しており，同じ文字種の文字は同じクラスに属す．提案手法では表 3.1に例示するような文字クラスを定義している．例えば，文字列 `img2019.jpg` が入力値である場合，以下の様な文字クラス列に変換される．

`img2019.jpg` → (`img`, 2019, `.`, `jpg`) → `AN.A`

文字列を文字クラス列に変換する仕組みを Algorithm 1に示す．まず，説明に用いる記号を定義する．ccmap はあらかじめ与えられる文字クラスの定義である．x[0:i]は文字列 x の 0 文字目から i 文字目の部分文字列を意味している．len(x)は x の長さを返す関数である．変換処理は 3 つの関数からなり，まずこの 3 つの関数 PREFIX-MATCH，GET-BEST-CANDIDATE，GET-CC-SEQ を定義する．関数 PREFIX-MATCH は入力文字列とある文字クラスを引数として，文字列を先頭から順にその文字クラスに包含される最長の文字列を見つけ出す処理を行う．関数 GET-BEST-CANDIDATE はあらかじめ定義されている全ての文字クラスに対して関数 PREFIX-MATCH を実施し，関数 PREFIX-MATCH の結果が最長となる文字クラスを

表3.1: 文字クラスの定義例

文字クラス	説明	例
A	アルファベット	{a, ..., z, A, ..., Z}
N	数字	{0, 1, ..., 9}
T	空白およびタブ	{␣, \t}
Q	クオート	{', "}
B	括弧	{(,), ...}
M	マルチバイト文字	{\x0100, ... }
.	ピリオド	{.}
/	スラッシュ	{/}

選ぶ処理を行う．関数 GET-CC-SEQ は関数 GET-BEST-CANDIDATE を入力値の文字列に対して複数回適用し，入力値の文字クラス列を得る処理を行う．

T3: Filtering

異常検知において，学習データには実際の通信を利用することが一般的であり，その中には攻撃を意図したものが含まれている可能性がある．攻撃であるデータを学習してしまうことは検知漏れを発生させる原因となってしまう．そこで，前ステップである要素について得た複数の文字クラス列に対してヒストグラムを作成し，ある文字クラス列の出現確率が極めて低い場合，これを除外する処理を行う．これは一般的な Web サイトでは実際の通信に含まれる攻撃は正常な通信に対して極めて少ない状況であることに基づいている．具体的にはある文字クラス列の出現確率が閾値 α $(0 \leq \alpha \leq 1)$ 未満の場合，これを除外する．例えば，ある URL パラメタの入力値が `img2019.jpg`，`img2020.png`，`../../etc/passwd` の 3 つである場合，文字クラス `AN.A` の出現確率は $\frac{2}{3} = 0.66$，文字クラス `././A/A` の出現確率は $\frac{1}{3} = 0.33$ となる．閾値 α を 0.4 とした場合，文字クラス `././A/A` が除外される．この処理により学習時にノイズとなるような攻撃データを学習しないようにし，検知漏れの発生を防ぐ．

T4: Selecting

入力値の中にはブログ記事のタイトル名やコメント欄のように，形式が定まっていない自由度の高い入力値も存在する．そのような入力値について文字クラス列のように制約の強い形式を当てはめると誤検知が生じてしまう原因となる．提案手法ではさらに文字クラス集合という概念を導入する．文字クラス集合とは文字クラス列に出現する文字クラスを抽出して重複を排除した集合であり，文字クラス列のように順序性を保持しない．提案手法ではプロファイルの

Algorithm 1 Character Class Sequence Generation

```
Require:
    ccmap = {A: abcd, N: 0123, ...}
 1: function PREFIX-MATCH(x, cc)
 2:     for i, y in x do
 3:         if y not in cc then
 4:             return i, x[0:i]
 5:         end if
 6:     end for
 7:     return len(x), x
 8: end function
 9:
10: function GET-BEST-CANDIDATE(cands)
11:     bc, bl = null, 0
12:     for c, l in cands do
13:         if l > bl then
14:             bc, bl = c, l
15:         end if
16:     end for
17:     return bc, bl
18: end function
19:
20: function GET-CC-SEQ(x)
21:     ccseq = []
22:     while len(x) > 0 do
23:         cands = []
24:         for c, cc in ccmap do
25:             l, z = PREFIX-MATCH(x, cc)
26:             if l > 0 then
27:                 cands += [(c, l)]
28:             end if
29:         end for
30:         bc, bl = GET-BEST-CANDIDATE(cands)
31:         ccseq += [bc]
32:         if bl == 0 then
33:             x = x[bl+1:]
34:         else
35:             x = x[bl:]
36:         end if
37:     end while
38:     return ccseq
39: end function
```

特徴として文字クラス列と文字クラス集合のいずれを用いるかを選択する．自由度の高い入力値の場合，順序に制約を持たない文字クラス集合を適用し，型が決まった入力値の場合，順序に制約をもつ文字クラス列を適用することで，誤検知を低下させつつ，検知漏れが発生することを防ぐ．具体的には，ある要素に対して，前ステップまでに得られた文字クラス列の集合に含まれるユニークな文字クラス列の個数を n とし，閾値 β と比較する．$n < \beta$ の場合，入力値は規則性を有すると見なし，全ての入力値から得られた文字クラス列をプロファイルとする．$n \geq \beta$ の場合，入力値は規則性をもたないものと見なし，得られた文字クラス列から文字クラス集合を求め，プロファイルとする．例えば，入力値から得られた複数の文字クラス列が (`AN.A`, `ANA.A`, `A/A.A`) とすると，文字クラス集合は {`AN./`} となる．

T5: Profiling

提案手法では Key-Value 形式でプロファイルを作成する．プロファイルの Key は HTTP リクエストの各要素の Key に相当する．URL パラメタやヘッダ，Cookie などはすでに Key-Value の形式になっているため，各要素の Key をそのままプロファイルの Key として扱う．また，URL パスや各要素の Key の部分に攻撃コードが挿入される可能性もあるため，URL パス，URL パラメタの Key，ヘッダの Key，Cookie の Key の値を Value として捉え，それぞれ個別のプロファイルを作成する．よって上記の例の HTTP リクエストによって作成されるプロファイル全体 P は以下のようになる．$p(x)$ は入力値 x で作成されたプロファイルを返す関数とする．

$$
P = \begin{cases}
\text{path} & = p(\texttt{/index.php}) \\
\text{parameter key} & = p(\texttt{id}, \texttt{name}) \\
\text{header key} & = p(\texttt{Host}, \texttt{User-Agent}) \\
\text{cookie key} & = p(\texttt{PHPSESSID}) \\
\text{parameters} & = \begin{cases} \texttt{id} & = p(\texttt{1}) \\ \texttt{name} & = p(\texttt{alice}) \end{cases} \\
\text{headers} & = \begin{cases} \texttt{Host} & = p(\texttt{example.net}) \\ \texttt{User-Agent} & = p(\texttt{Mozilla/5.0}) \end{cases} \\
\text{cookies} & = \begin{cases} \texttt{PHPSESSID} & = p(\texttt{1234}) \end{cases}
\end{cases}
$$

3.3.2 検知

検知には 3 つのステップがあるが，最初の 2 つのステップ **D1**，**D2** は学習時のステップ **T1**，**T2** と同じである．つまり，学習時と同様に検知対象の入力値を文字クラス列へ変換する．その後，**Step D3 Comparison** にてその文字クラス列とプロファイルの類似度を求める．類似度の求め方は学習したプロファイルの特徴の種別によって異なる．

プロファイルが文字クラス列の場合

プロファイルに含まれる n 個のユニークな文字クラス列，それぞれについて検知対象の文字クラス列との最長共通部分列 (LCS) [42] を求め，類似度を算出する．類似度 s は 2 つの文字クラス列を x，y とすると，以下の式で定義される．

$$
s = \frac{|LCS(x,y)|}{|x| + |y| - |LCS(x,y)|}
$$

例えば 2 つの文字クラス列 `AN.A` と `ANA.A` の類似度は $\frac{4}{4+5-4} = 0.8$ となる．算出した類似度の集合を $S = \{s_1, s_2, ..., s_n\}$ とする．次に，得られた複数の類似度 S の中で最大の値 $max(S)$ を求め，異常値 $a = 1 - max(S)$ と閾値 γ $(0 \leq \gamma \leq 1)$ を比較する．$a \geq \gamma$ の場合，プロファ

イルとの乖離が大きいため，攻撃と判定する．$a < \gamma$ の場合，プロファイルと類似しているため，正常と判定する．

プロファイルが文字クラス集合

プロファイルの文字クラス集合を E_p，検知対象の文字クラス列から得られる文字クラス集合を E_t とすると

$$E_t \subseteq E_p$$

である場合，正常と判定し，そうでなければ攻撃と判定する．文字クラス集合は文字クラス列に比べ，制約条件が少ないため，検知漏れを発生させる可能性が高い．そのため，学習に出現しない文字クラスが 1 つでも発生した場合に攻撃として検知するという厳しい条件を採用している．

3.4　評価

提案手法を実装し，実データセットを利用した評価を行った．データセット，実験環境，評価指標および評価結果を各節にて述べる．

3.4.1　データセット

著者らが外部に公開している複数の Web サーバで観測した HTTP リクエストをデータセットとして用いた．評価はホールドアウト法 [43] を用いて行った．Web サーバごとに学習データ数: 検知データ数を 1:2 の割合で用意し，学習データから HTTP レスポンスのステータスコードが範囲 [200, 400) 以外となるデータおよび，攻撃とラベルづけされたデータを除外した．

学習データが検知データより少ない理由は Web サイトの性質を考慮したためである．Web サイトは更新や仕様変更が行われる可能性が高く，更新や仕様変更が行われた際はプロファイル等を再学習する必要がある．そのため，学習データが多くない状態を想定した．評価にホールドアウト法を選択した理由は，実際の Web サイト更新や仕様変更を踏まえた評価を行うためである．本来，Web サイトの更新や仕様変更があると検知精度は低下することが考えられる．クロスバリデーション法のようにデータ全体を分割しながら検証を行う方式だと，Web サイトの更新や仕様変更という時系列変化を無視してしまうため，実際よりも検知精度が向上してしまう可能性がある．そこで，本評価では，学習データと検知データが時系列順になるようにした．ステータスコードによる除外を行った理由はこのような HTTP リクエストは脆弱な Web アプリケーションが稼働していないかを調査するスキャンの可能性が高く，学習すべき正常なデータではないと判断したためである．

学習データと検知データについては，攻撃か否かを示すラベルを付与した．ラベル付与では

表3.2: データセットの概要

サイト名	学習データ数	検知データ数		
		攻撃数	正常数	合計
A	3,534	237	11,498	11,735
B	8,658	311	20,426	20,737
C	10,977	414	30,284	30,698
D	6,454	281	31,820	32,101
E	15,948	233	37,446	37,679
F	36,078	176	90,353	90,529
G	20,653	284	99,469	99,753
H	45,006	4,298	123,927	128,225
I	59,390	2,468	147,085	149,553
J	40,384	140	154,633	154,773
K	51,314	180	163,212	163,392
L	67,659	1,074	197,285	198,359
M	88,222	148	226,061	226,209
合計	454,277	10,244	1,333,499	1,343,743

OSS の IDS/WAF である ModSecurity [44] や Snort [45]，フリーで利用できる Emergence Threats [46] のシグネチャによって攻撃か否かを判断した結果を元に，一部人手でチューニングを行った．

表 3.2に評価に使用したデータセットの概要を示す．13 の Web サイトから合計 1,798,020 件の HTTP リクエストを収集し，454,277 件を学習データ，1,343,743 件を検知データとした．収集期間は 2017 年 1 月 1 日から 2017 年 1 月 31 日である．検知データに含まれる攻撃種別の割合は図 3.2に示す通りである．凡例の SQLI は SQL インジェクション，PT はパストラバーサル，RCE はリモートコード実行，RFI はリモートファイルインクルージョンを表している．複数種類の攻撃が含まれており，評価に妥当なデータセットである．

検知データには学習時には出現しなかった Key が出現することがある．このような Key については学習時にプロファイルが作成されていない．検知時においてこのような未学習な Key に関する扱い方には検討の余地があるが，今回の実験では未学習の Key が発生した場合は異常と判定するには根拠が不足していると考え，正常であると判定することとした．例えば，学習によって URL パラメタに `id`，`name` のプロファイルが作成されたとする．検知時に `page` という URL パラメタが出現した場合，異常と判定できないため正常であると判定することと

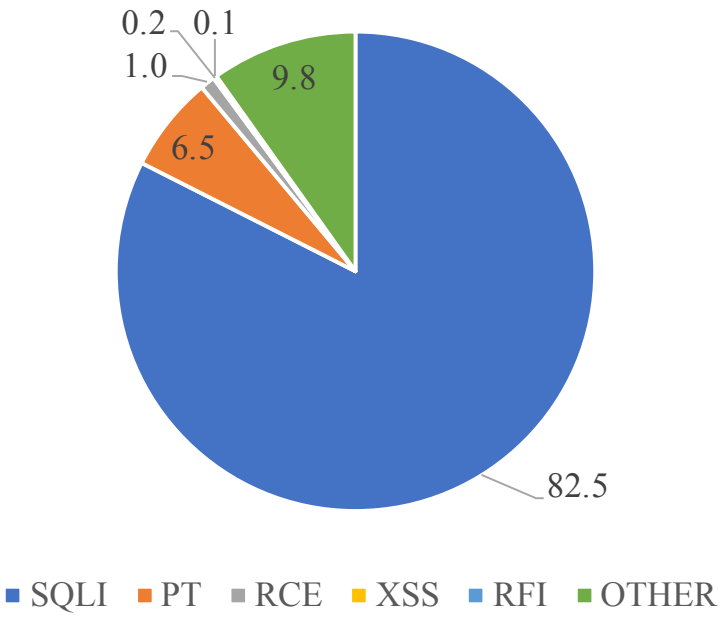

図3.2: 検知データに含まれる攻撃種別の割合 (%)

する.

3.4.2 実験環境

評価に際して、提案手法および既存手法の実装を Python 言語で行った．実行に際しては Python インタープリタより高速な実行環境である PyPy [47] を利用した．また実験に使用したマシンのスペックは 3.1GHz Intel Core i5 CPU，16GB RAM, 1TB SSD であった.

3.4.3 指標

既存手法と提案手法について検知率 (True Positive Rate)，誤検知率 (False Positive Rate) および処理時間，それぞれの観点で比較した．検知率，誤検知率はそれぞれ以下のように定義した.

$$\text{検知率} = \frac{\text{攻撃データを攻撃として判定した数}}{\text{攻撃データ数}}$$

$$\text{誤検知率} = \frac{\text{正常データを攻撃として判定した数}}{\text{正常データ数}}$$

3.4.4 検知精度

図 3.3に提案手法および既存手法の ROC 曲線 [48] を示す．ROC 曲線は横軸に誤検知率，縦軸に検知率をとった上で，各手法における閾値を調整した際の誤検知率，検知率の点をプロットし，それを線で結合した曲線である．より左上にある曲線の方がより精度が高いことを表している．攻撃検知の分野では正常リクエストが攻撃リクエストより著しく多いことが一般的であるため，高い誤検知率は許容できない．そのため，本研究では誤検知率の許容範囲を 10% までとしている．

既存手法については論文をもとに著者らが実装を行った．Anagram については n-gram の値 n，Luo らの手法では XGBoost の学習繰返し回数の閾値 b，Betarte らの手法では初期クラスタ数 c，また全ての手法において検知時に攻撃として判定する閾値 t が検知精度に対して大きな変化をもたらすと考え，これらのパラメタを調整することで図 3.3を求め，最適な検知精度を調査した．より具体的には，$n = [1, 10]$，$b = [10, 1000]$，$c = [1, 500]$ および $t = [0.0, 1.0]$ の範囲内で閾値を変化させた．Luo らの手法で利用する XGBoost は教師あり学習のため，Luo らの手法の評価時のみ学習データに含まれる攻撃ラベルのデータを除外せずに学習を行った．

検知率および誤検知率は Web サイト間で異なるため，Web サイト間のデータ量の差異を考慮し，ROC 曲線を描画する際には各 Web サイトの検知率および誤検知率の加重平均を用いた．つまり，各 Web サイトの検知率および誤検知率を求めた後，検知率の平均値および誤検知率の平均値を ROC 曲線の検知率，誤検知率とした．図 3.3より既存手法に比べて提案手法はより低い誤検知率で高い検知率が達成できており，検知精度において既存手法より高い性能であることが示された．Luo らの手法および Betarte らの手法の検知率が特に低い理由は特徴量の取り方が問題であった．Luo らの手法では HTTP リクエスト全体のバイト値を特徴ベクトルとするため，URL 長の変化やヘッダ長の変化が大きいサイトに対しては特徴ベクトルの要素の次元にずれが発生するため高い精度を発揮できないと考える．Betarte らの手法ではあらかじめ用意した攻撃の特徴を示す文字列の出現数を特徴ベクトルとするため，攻撃にこれらの文字列が出現しない場合に検知率は大幅に低減してしまう．

Web サイトごとに検知率，誤検知率にはばらつきがある．表 3.3に Web サイト間の検知率／誤検知率の平均および標準偏差を示す．ROC 曲線より得られる各手法にて最大の精度を発揮する閾値を設定した際の結果を用いている．提案手法の検知率の標準偏差 (14.8) および誤検知率の標準偏差 (0.25) は既存手法と比較して小さく，各 Web サイトに対して安定的に高い精度維持できることを示している．また，既存手法に比べ，閾値調整を繊細に行わなくても，一定の精度を発揮することが可能であることから，新しい Web サイトを検知対象に追加する際も運用者にとって手間が少ないという効果をもたらす．

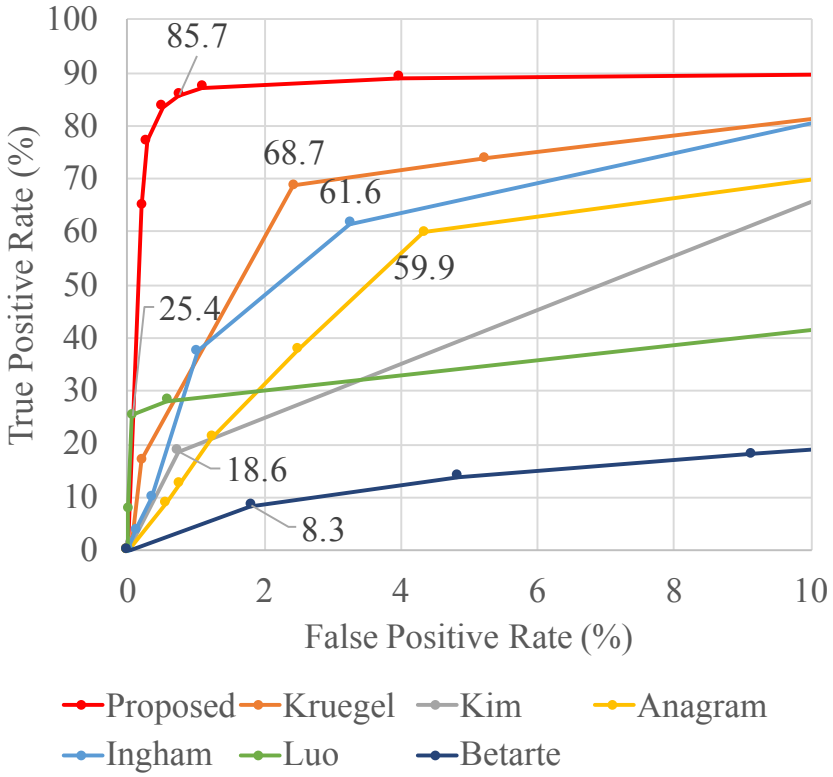

図3.3: ROC 曲線

表3.3: Web サイト間の検知率／誤検知率の誤差

手法	検知率 (%)		誤検知率 (%)	
	平均	標準偏差	平均	標準偏差
Proposed	83.5	14.8	0.50	0.25
Kruegel	68.7	22.9	2.43	0.99
Kim	18.0	25.7	0.73	2.54
Angram	59.9	25.4	4.33	2.54
Ingham	60.3	27.8	3.25	2.47
Luo	25.4	31.6	0.07	0.11
Betarte	8.13	10.5	0.92	0.56

3.4.5 閾値分析

提案手法には学習時のステップ **T3** において，出現率が少ない文字クラス列を除外するために利用する閾値 α，学習時のステップ **T4** において，順序制約を持つ文字クラス列または順序

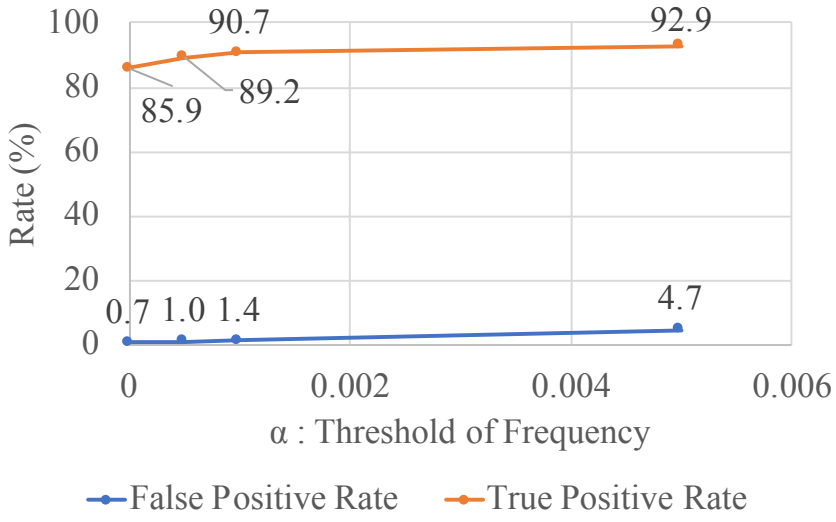

図3.4: 閾値 α の変化 ($\beta = 10, \gamma = 0.5$)

制約を持たない文字クラス集合のどちらをプロファイルに用いるかを判定する閾値 β，および検知時のステップ **D3** に入力値とプロファイルとの類似度に対して攻撃か否かを判断する閾値 γ という 3 つの閾値が存在する．それぞれの閾値について値を変化させた場合の検知率と誤検知率の変化に対する分析を行った．

閾値 α に関する分析

閾値の変化にともなう検知率，誤検知率の変化を図 3.4に示す．横軸は閾値のとる値を表しており，縦軸は検知率／誤検知率を表している．α を大きくすると検知率，誤検知率はともに単調増加することが分かる．この原因は α を大きく取った場合，学習時に含まれる出現確率が低いであろう攻撃データを除外するとともに，正常なデータまでも除外してしまうことにある．また，α の値が $[0, 0.001]$ 区間と小さい場合，誤検知率の増加 (0.70%) より検知率の増加が大きい (4.79%) が，α の値が $[0.001, 0.005]$ 区間と大きい場合，誤検知率の増加 (3.3%) が検知率の増加 (2.2%) よりも大きくなるため，誤検知を多発させないためにも，α は 0 に近い値に設定するのが望ましいと考える．

閾値 β に関する分析

図 3.5により β の変化に対して誤検知率は大きく変化しないことがわかる．また，検知率にはピークが存在し，今回の実験では $\beta = 10$ 付近であると推測できる．β の値が小さいとき，文字クラス集合がプロファイルの特徴として選択される割合が高くなるため順序性の制約条件が緩和された分，検知率も低い．β の値が大きいとき，文字クラス例がよりプロファイルの特徴として選択される割合が高くなりすぎるため，検知率向上に寄与していた文字クラスによる

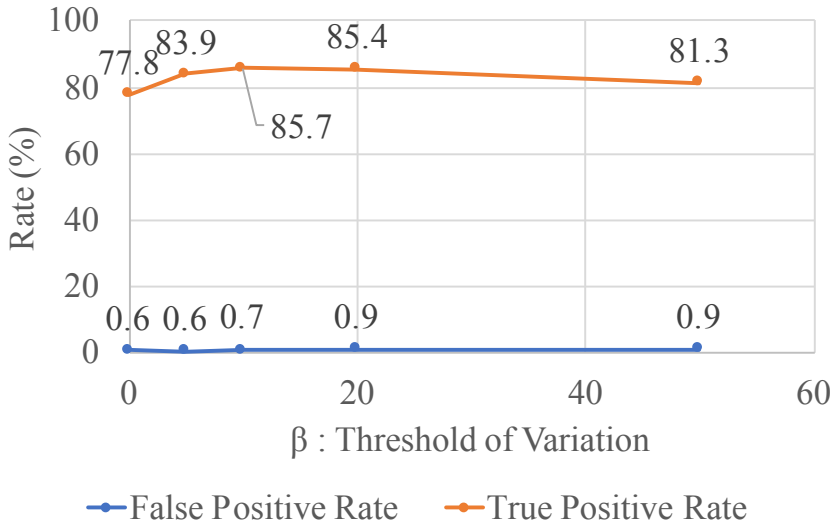

図3.5: 閾値 β の変化 $(\alpha = 0.001, \gamma = 0.5)$

制約が取り除かれてしまい，検知率が低下した．例えば，入力値 `hello 2019/03` を学習して，SQL インジェクションである文字列 `or 1=1` を検知することを考える．文字クラス集合として学習する場合，`{ATN/}` となるため，検知時，文字 `=` が学習済みの文字クラスに含まれないため，攻撃として検知できる．しかし，文字クラス列として強制的に学習させた場合，文字クラス列 `ATN/N` となり，`or 1=1` の文字クラス列 `ATN=N` との類似度は $\frac{4}{5+5-4} = 0.66$ と高い値となるため，検知率が低下してしまったことが原因である．

閾値 γ に関する分析

図 3.6より，γ が増加すると，検知率，誤検知率は単調減少する．γ の増加に対して誤検知率より検知率の方が減少が著しいため，α と同様，0 に近い値に設定することが望ましい．

3.4.6　誤検知分析

検知データ全体に対して提案手法の精度が最も高くなる閾値設定である $\alpha = 0.001, \beta = 10, \gamma = 0.5$ とした場合，提案手法では 5,296 件の誤検知が発生していた．図 3.7に HTTP リクエストの要素ごとの誤検知の件数を示す．誤検知の多くはヘッダおよび Cookie であり，URL パスや URL パラメタと比較すると入力値の自由度の高さが誤検知の原因になっていると考える．

誤検知の要因は学習時に現れなかった文字列の出現や，文字列の構造の変化である．実験で確認できた主な誤検知の事例を紹介する．

XFF ヘッダに起因する誤検知　XFF (X-FORWARDED-FOR) ヘッダはプロキシ等を経由し

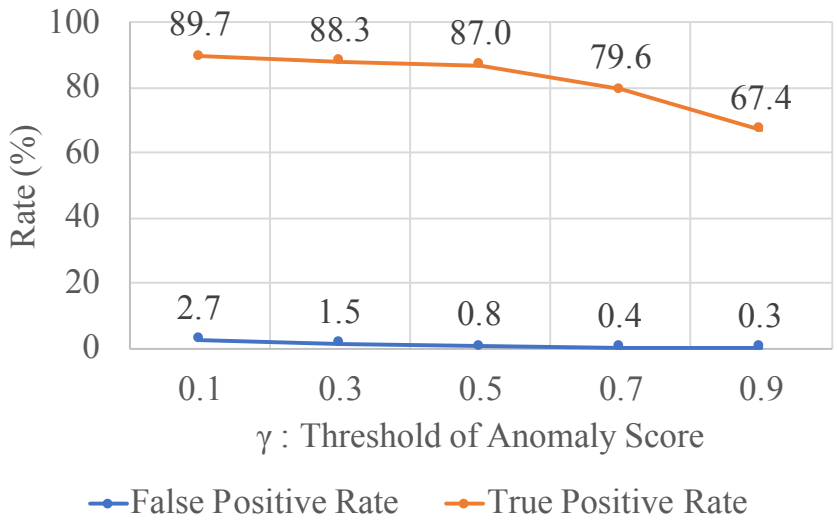

図3.6: 閾値 γ の変化 $(\alpha = 0.001, \beta = 10)$

て HTTP リクエストが送信された場合にその送信元を示している．実験では学習時は IPv4 アドレスのみが観測されたため，文字クラス列 `N.N.N.N` といったプロファイルが作成されたが，検知時は `unknown` といった値が観測され，文字クラス列が `A` となり，誤検知となっている．この事象はプロキシサーバの設定の違いによって発生する．

Base64 エンコード文字列に起因する誤検知　HTTP 1.X の場合 Web アプリケーションはバイナリ等のデータを HTTP リクエストにて送信する際，Base64 エンコードして ASCII 文字に変換してから送信する場合が一般的である．提案手法では Base64 エンコードという型を識別できる訳ではないため，アルファベット，数字などから成る文字クラスの集合として認識する．よく利用される RFC 4648 の Base64 エンコーディングではアルファベット，数字および記号 `+`，`/` の 64 文字で構成される．そのため記号 `+`，`/` は $\frac{2}{64} = 3.13\%$ の確率でしか出現しないため，学習時に観測できず，検知時に誤検知を発生させる原因となる．

これらの要因による誤検知は学習時に特徴を工夫して解決することは困難であるため，関連研究にて述べる学習方法の工夫により解決を図ることが可能である．例えば，XFF ヘッダの場合，1 つの Web サイトの観測データだけでなく，複数の Web サイトの観測データから学習し，プロファイルを共有することで誤検知を減らす可能性を上げることができる．

3.4.7　処理時間

図 3.8と図 3.9に，各 Web サイトにおいて学習と検知に要した処理時間を示す．各手法で検知精度が最も高くなるパラメタの場合を計測した．横軸は各 Web サイトの処理データ数であ

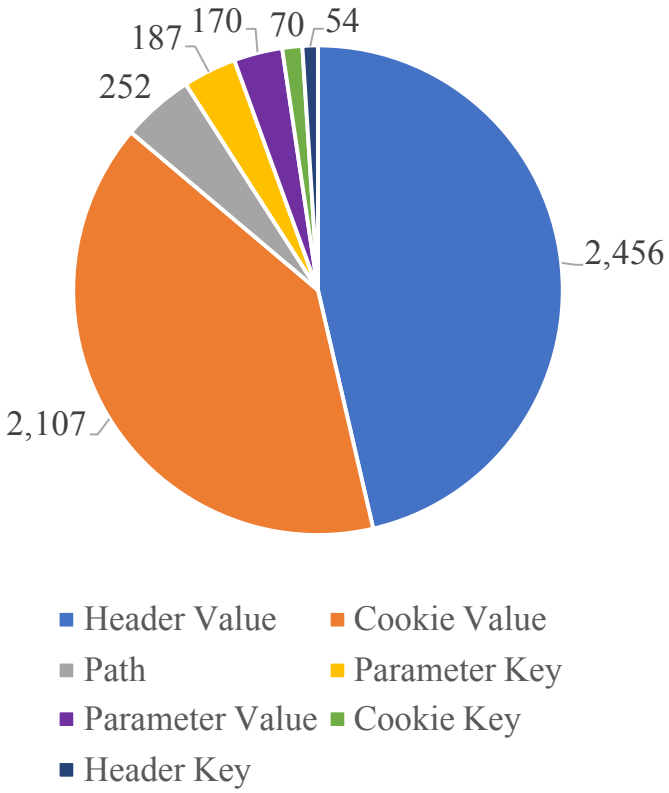

図3.7: 要素ごとの誤検知の件数

り，縦軸は処理時間 (秒) である．破線は各手法の結果の近似線である．近似線より，提案手法の学習時の処理速度は約 1.01 ミリ秒/件，検知時の処理速度は約 0.49 ミリ秒/件であった．既存手法と比較すると，同等ぐらいの処理速度であった．

提案手法はオフラインでの学習を想定しているが，オンラインでの学習を適用した場合でも，秒間約 1,000 件のアクセスが行われる大規模サイトにおいてもリアルタイムな処理が可能であることが期待できる．Web サイトの死活監視サービスを行っている pingdom [49] によると Web ページあたりの平均的な HTTP リクエスト数は 110 件ぐらいであるため，検知時に提案手法をインライン[*1] で適用した場合においても，約 54.1 ミリ秒の遅延に抑えることが可能である．同じく Pingdom の調査によると，Web ページの遅延時間は約 3 秒までが理想的とあり，3 秒に対して，54.1 ミリ秒は 1.8% であり，ごくわずかな増加であるので，処理速度面においては実用的であると考える．

処理データ数と処理時間が比例しない箇所が存在する．これは処理時間が HTTP リクエスト数のみならず，各 HTTP リクエストに含まれる URL パラメタの数やヘッダの数および入力値の複雑性にも左右されるためである．

[*1] クライアントーサーバ間に適用し，攻撃と検知した場合遮断できるようにする実施方法

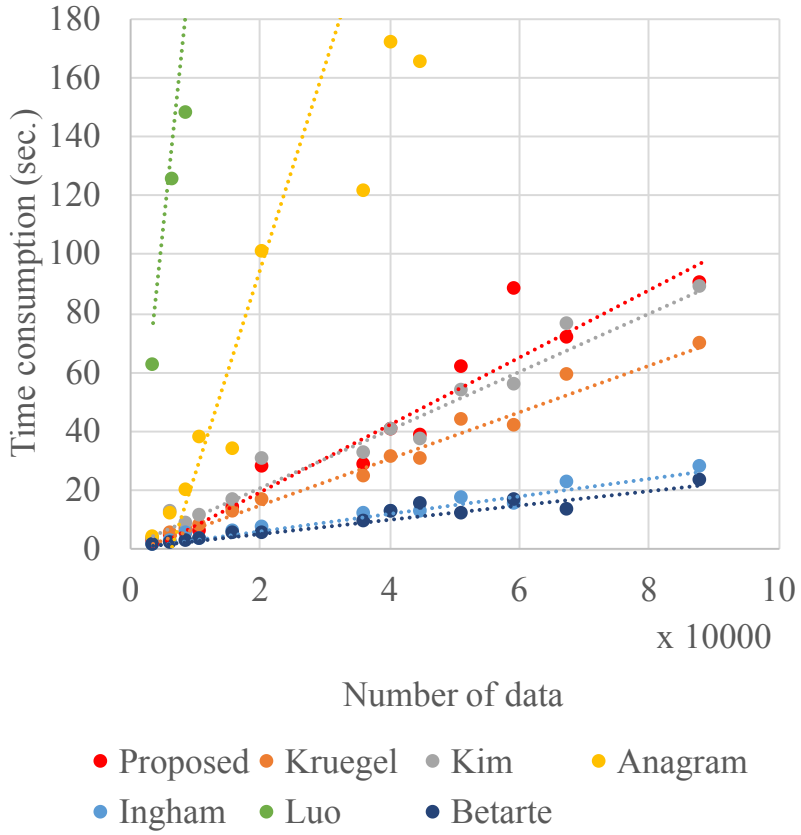

図3.8: 学習時の処理時間

3.5 関連研究

ネットワーク通信に対する異常検知を適用する手法 [28, 29, 30] では通信プロトコルに依存せずに対応する必要があるため，ネットワーク通信に現れるバイト列の頻度あるいは，通信の特徴を数値化した後に異常検知を適用することが一般的である．

Web アプリケーションの通信に特化した異常検知手法には古くから文字列長や n-gram 等を利用する手法 [1, 3, 4, 5] などが存在する．近年では，並列的に学習器を 1 つずつ順番に構築していく XGBoost を利用する Luo らの手法 [6] や，混合ガウスモデルを利用した Betarte らの手法 [7] が存在する．これらの手法では HTTP リクエストを特徴ベクトルに変換する際に工夫がないと高精度な検知は期待できない．提案手法では比較的容易である文字クラス列をあらかじめ定義すればよいため，特徴量の調整が少なく安定した検知精度を発揮できると考える．近年ではニューラルネットワークを活用した深層学習に基づく異常検知手法も提案されている [50, 51, 52]．ニューラルネットワークを活用する場合，検知精度を維持するために特徴量を選択したり調整したりする必要がない点が利用者にとっての利点である．例えば，本手法

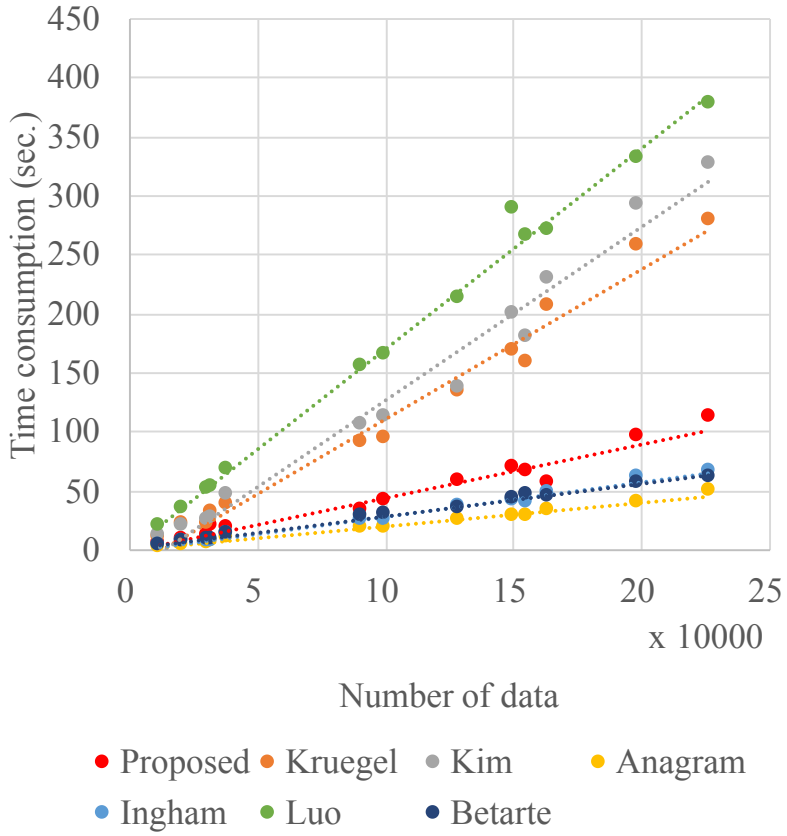

図3.9: 検知時の処理時間

では利用者があらかじめ文字クラスを定義する必要があるが，そういった定義が事前に必要なくなる．その反面，何を根拠に判定しているかは説明ができていないのが弱点である．

3.2節で述べた既存手法は特徴の捉え方を工夫することで検知精度を向上させるものである．しかし，特徴の捉え方以外にも学習方法を工夫することで誤検知を低減させる手法が存在する．Maggi らは Web アプリケーションの仕様が変化して，URL パラメタの入力値の仕様が変化した場合に誤検知してしまう問題に着目し，HTTP レスポンスの内容を活用して，逐次的に再学習を行い，プロファイルを更新することで誤検知を低減する手法 [41] を提案している．Robertson らは，ある URL パラメタに対する学習データが少ない場合，検知時は誤検知が多発してしまう問題に着目し，ある URL パラメタについて正確なプロファイルを作れない場合でも，他の正確に学習できた URL パラメタのプロファイルを活用することで，誤検知を低減する手法 [53] を提案している．提案手法においてもこれらの学習方法を工夫する手法と組み合わせることで更に精度を向上させることが可能である．

3.6 結語

本稿では Web アプリケーションの脆弱性を悪用する未知攻撃を検知する際に発生する誤検知の課題に対して，HTTP リクエストの各要素の文字列の構造に着目し，学習を行うことで，誤検知を低減する手法を提案した．評価の結果，既存の異常検知手法に比べ提案手法の方が精度が高く，また処理性能も実用的な値であることを示した．今後は，入力値が学習時に出現する頻度が低い場合，出現しない場合の対応策を検討し，更なる誤検知の低減を目指す．

第 4 章

HIDS アラートと HTTP リクエストの関連付け手法

4.1 緒言

攻撃検知に用いられる IDS はその利用形態から大きくネットワーク型 (NIDS) およびホスト型 (HIDS) の 2 つに大別される．本研究では NIDS はネットワーク通信に発生するリクエストやレスポンスのバイト列を特徴量として攻撃検知を行うものであり，HIDS は Web アプリケーションが動作する Web サーバ内で OS やアプリケーションから発生するシステムコールやイベントを特徴量として攻撃検知を行うものとして考える．

NIDS の問題はアラート数が膨大になってしまうことである．この原因は 2 つあり，1 つ目は NIDS と HIDS に共通する課題であるが，正常アクセスに対する誤検知である．誤検知を低減する手法 [2, 32, 53, 54] が古くから多くの研究者によって提案されてきており，本研究においても 3章において誤検知を低減する手法を提案した．2 つ目は攻撃ではあるが失敗した攻撃の検知である．一般に NIDS は攻撃の成否を判定することができないため，失敗した攻撃の検知を減らすことは困難である．

一方，HIDS はホスト内の情報を活用して攻撃検知を行うシステムである．HIDS は攻撃が成功してサーバに対して影響を与えている状態を異常として捉えることができる．特に，改ざんを行う攻撃はネットワーク通信に痕跡が現れないため，HIDS によるアプローチが必要となる．HIDS には，ファイルの完全性を確認することで改ざんを検知する仕組みなどホスト内の様々な情報を活用するものがあるが [55]，本稿では，OS のシステムコールや DB に対する SQL クエリ発行の情報をもとに攻撃検知を行う HIDS を対象として考える．

HIDS がアラートを通知すると，管理者は攻撃による被害有無の確認や被害が再発しないよう暫定対処を行うための情報を収集する調査を行う．ここでいう暫定対処とは例えば，攻撃の標的となった Web アプリケーションの URL に対するアクセスを禁止したり，攻撃の送信元

となっている IP アドレスを遮断したりなど根本的な解決ではないが，一時的に同じ攻撃によって再度被害が発生しないようにするための処置である．そのため，アラートの調査では標的アプリケーションの URL や送信元 IP アドレスなどの情報を含む HTTP リクエストの情報が必要である．HIDS ではシステムコールや SQL クエリを始めとするホストに関する情報を入力とするため，改ざんされたファイルや実行されたコマンド等の被害については出力可能であるが，対象の Web アプリケーションの URL や送信元 IP アドレスなどの情報を出力できない．そのため，管理者はこれらの情報を HIDS の検知結果から人手で特定する必要があるため，調査に時間を要してしまう．

本研究では HIDS の入力であるシステムコールや SQL クエリ発行に対して，それらを発生させた HTTP リクエストを処理したスレッドの ID と高精度な処理開始および終了時刻に基づいて HTTP リクエストを関連付ける手法を提案する．関連付けにより，HIDS が異常と検知したシステムコールや SQL クエリ発行がどの HTTP リクエストによって発生したかを特定できるようになる．そのため，管理者は HIDS の結果から被害の原因となった Web アプリケーションに関する情報は把握することができる．

本研究の貢献は以下の通りである．

- HIDS の入力であるシステムコールや SQL クエリ発行に対して HTTP リクエスト情報を関連付けることで，HIDS が検知した際に標的となった Web アプリケーションや攻撃の送信元を特定する手法を実現した．
- 様々な Web アプリケーションに対して関連付け精度および処理速度の評価を行い，誤った関連付けがないこと，および Web アプリケーションのパフォーマンス低下を平均 5% 程度に抑えられることを示した．

4.2　既存のイベント関連付け手法とその課題

システムコールや SQL クエリ発行の情報に HTTP リクエストを関連付けることは異なる種類のイベントを関連付けることと考えることができる．異なる種類のイベントを関連付ける手法にはイベントの発生時刻の近さやイベントを発生させたプロセスの ID (PID) あるいはスレッドの ID (TID) 等の識別子の関係を利用して関連付ける手法が提案されてきた．Skopik らの手法 [56] は一定の時間間隔を予め定義し，その間隔内に起きたイベントを関連付けるものである．しかし，適切な時間間隔を決めることは困難であり，定義した時間間隔が長すぎると関係のないイベントを関連付けてしまったり，時間間隔が短すぎると関係するイベントであるが関連付けられない事象が発生してしまうため正確性が不十分である．イベントに PID, TID や親プロセス ID (PPID) など明示的に関連付けることができる情報が含まれている場合，BackTracker [57] が利用できる．この手法では予め指定した期間に発生するイベントに

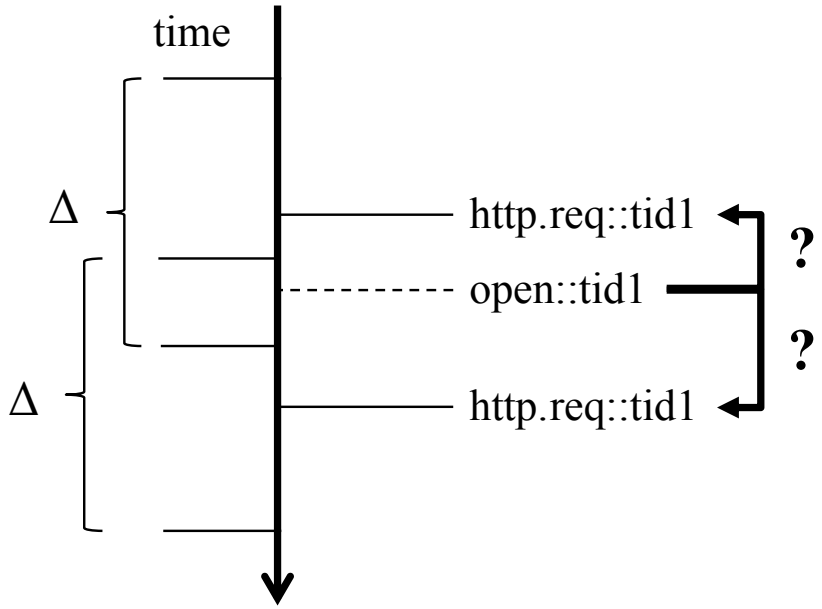

図4.1: PID/TID に基づく関連付けの課題

対して PID/TID などが一致するか否かで関連付けを行うため，時間間隔のみによる関連付け手法より正確である．例えば，HTTP リクエストが常に異なるプロセス／スレッドで処理される場合，HTTP リクエストと関連付けるべきイベントの PID/TID は一意に定まるため BackTracker による関連付けは正確である．しかし，サーバプロセスのような同じプロセス／スレッドで異なる HTTP リクエストを処理する場合，Skopik らの手法と同様な課題が発生する．図 4.1にこの問題の様子を示す．2 つの HTTP リクエストがスレッド 1 によって処理され，その間にスレッド 1 からファイルアクセスが発生した．Skopik らの手法と同様，BackTracker は PID/TID という関連付けする際の制約があるものの，関連付け対象となるイベントから時間間隔 Δ に含まれるイベントを関連付ける．そのため，図 4.1のような場合このファイルアクセスはどちらも HTTP リクエストの発生時刻から Δ 以内に含まれるため，どちらか断定することができないという課題が存在する．

本論文ではこれらの関連付けの正確性に関する課題を解決する関連付け手法を提案する．

4.3 提案手法

図 4.2に提案手法の概要を示す．提案手法は HTTP リクエストおよびイベントの収集 (Collecting) ，HTTP リクエストとイベントの関連付け (Event Correlating) およびイベントとアラートの関連付け (Alert Correlating) の 3 つのフェーズに大別される．提案手法では 2 つの HIDS の形態を想定している．図 4.2の HIDS (a) の様に HIDS が提案手法で収集した

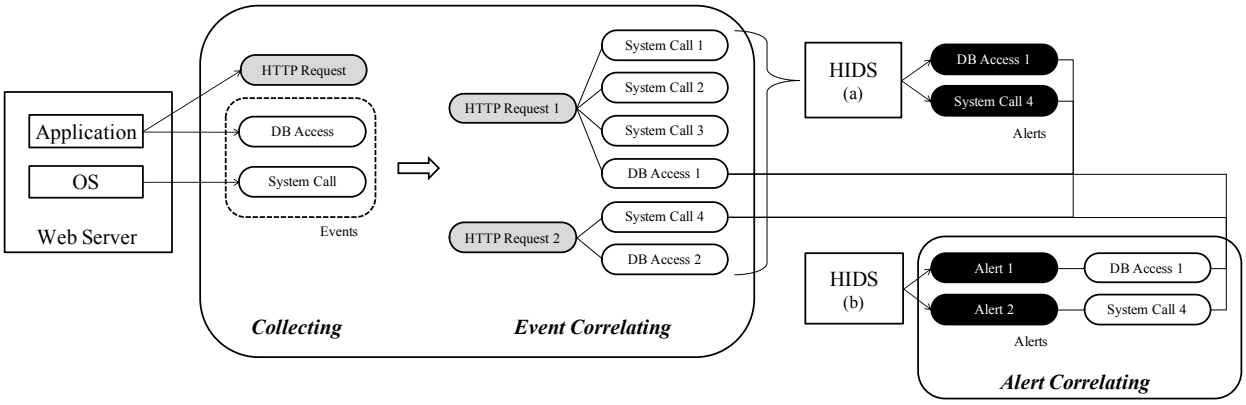

図4.2: 提案手法の概要

イベントを入力として，アラートを出力する際にそのイベントを出力できる形態および 図 4.2 の HIDS (b) の様に HIDS が提案手法で収集したイベントを入力とせず，出力するアラートとイベントを関連付ける必要がある形態である．

Collecting フェーズでは OS のカーネルやプロセスからシステムコールや SQL クエリ発行に関する関数呼出しの情報を収集する．本研究ではこれらの情報をイベントと呼び，システムコールや SQL クエリ発行の単位をイベントの単位として扱う．Event Correlating フェーズでは収集した複数のイベントがどの HTTP リクエストに関連するものかを複数の条件に基づき関連付けを行う．関連付けにより，ある HTTP リクエストがサーバ内のどのようなイベントを引き起こしたかが判別される．その結果，HIDS に提案手法で収集したイベントを入力とすることで，HIDS が異常なイベントを検知した際に，そのイベントがどの HTTP リクエストに関連付いているものかを特定することが可能となる．Alert Correlating フェーズではアラートとイベントの関連付けを行う．この処理は HIDS が図 4.2の HIDS (b) の形態の場合のみに実施する．

図 4.3に提案手法が利用者に対して提示する情報のイメージを示す. 提案手法の出力は HIDS のアラート，関連付いた HTTP リクエスト，および関連付いたイベントの 3 つの情報である．4 から 8 行目は HIDS アラートに関する情報であり，3 つのイベントが HIDS によって異常と検知されたことを示す．10 から 12 行目は HIDS アラートが示すシステムコールを引き起こした HTTP リクエストの内容を示している．14 から 20 行目は上記 HTTP リクエストが引き起こした全てのイベントを示している．これらの情報が関連付けされていることにより，利用者は HIDS アラートの分析の際に被害発生の原因となる HTTP リクエストを自ら関連付ける必要がなくなり，分析時間が短縮できると考える．

```
1  ================================================
2  * Report of proposed system #1
3  ================================================
4  * HIDS alert:
5    * anomaly events:
6      - execve, sed -ie s/deny/allow/g .htaccess
7      - open,   ./sed3wZccC, O_RDWR|O_CREAT
8      - rename, ./sed3wZccC, .htaccess
9  ------------------------------------------------
10 * correlated http request:
11   - POST /user/register?
12     element_parents=account/mail/%23value
13 ------------------------------------------------
14 * correlated host events:
15   - syscall, open, .htaccess, O_RDONLY:O_CLOEXEC
16   - syscall, connect, /var/lib/mysql/mysql.sock
17   - db,      SELECT cid, data, ... FROM ...
18   ...
19   - syscall, open, .htaccess, O_RDONLY
20   - syscall, open, ./sed3wZccC, O_RDWR,O_CREAT
21   - syscall, rename, ./sed3wZccC, .htaccess
22 ================================================
```

図4.3: 利用者に提示する情報の例

4.3.1 Event Correlating フェーズ

既存の関連付け手法 [56, 57] の問題は一定の時間間隔を定めて関連付けるため，関連付けが不正確になることにあった．そのため，本手法では既存手法が利用する PID/TID 等の識別子による区別に加え，予め定義された一定の時間間隔ではなく，HTTP リクエストの処理を踏まえて動的に決定される時間制約を設けることでイベントがどの HTTP リクエストによって発生したかを区別する．以下に関連付けを行う条件を示す．

条件 1 イベントを発したスレッドが HTTP リクエストを処理したスレッドと同一，あるいはイベントを発したプロセスが HTTP リクエストを処理したプロセスを根とするプロセスツリーに含まれる場合

条件 2 イベントの発生時刻が HTTP リクエストの処理スレッドによる処理開始時刻と処理終了時刻の間にある場合

条件 1 および**条件 2** 両方が満たされる場合に関連付けを行う．

条件 1 は既存手法 [57] と同様に PID や TID の一致に基づいた関連付けの条件である．Web サーバは一般に複数のプロセスを起動する，リクエストの処理を行う動作モデルには，各プロセス内に 1 つのスレッドを用意し待機する Prefork モード，各プロセス内に複数のスレッドを用意し待機する Worker モードの 2 つが存在する．いずれの動作モードであっても，1 つのスレッドは 1 つの HTTP リクエストの処理しか行わない．近年，これらの動作モデルよりリソ

ース活用が効率的な Event モードという動作モデルも利用されてきているが，待機プロセス／スレッドが同時に 2 つ以上のリクエストを処理することはないという点は Prefork/Worker モードと共通している．本手法では上記の様な 1 つのプロセスまたはスレッドは HTTP リクエストの処理を終えるまで他の HTTP リクエストの処理を行わない Web サーバを対象とする．

Web サーバは HTTP リクエストを受け取ったら，待機しているスレッドのどれかに HTTP リクエストを割り当て処理を行う [58]．この時，待機スレッドは HTTP リクエストを同時に 2 つ以上処理を行うことはないため，TID の一致という条件からイベントが HTTP リクエストに関連するか否かを見分けることができる．システムコールが OS コマンドの実行等，子プロセスを生成する場合，実行される OS コマンドは HTTP リクエストを処理しているスレッドとは異なるスレッドでの処理になる．そのため，プロセスの親子関係を加味した，プロセスツリーに基づく関連付けを行う．具体的には`fork`あるいは`clone`システムコールの戻り値が実行された OS コマンドを処理するプロセスの PID となるため，子プロセスが生成された際は，子プロセスから発生したイベントを親プロセスで処理している HTTP リクエストに関するイベントとして関連付ける．

条件 2 は提案手法独自の関連付けの条件である．スレッドによる HTTP リクエストの処理開始および処理終了時の高精度な時刻を取得し，あるイベントがどの HTTP リクエストの処理中に発生したかを判定する．これにより，同一の TID となるスレッドで処理される異なる HTTP リクエストに起因して発生したイベントを区別する．この条件は 1 つのスレッドが同時に 2 つ以上の HTTP リクエストを処理することがないという Web サーバの特性 [58] に基づいている．高精度な時刻情報をどのように取得するかは 4.3.3節にて詳説する．以下に 2 つの関連付けの例を示す．

図 4.4に 3 つの HTTP リクエストとそれに関連付けられたイベントの例を示す．3 つの HTTP リクエストはそれぞれスレッド 1, スレッド 2, スレッド 1 によって処理されている．この例ではイベントと HTTP リクエストの TID が一致したことにより関連付けられた様子を示している．図 4.4ではイベントの種類 (event type) とそのイベントの TID/PID/PPID を表すのに，`event_type::tid1` や`event_type::pid1` の様に表現する．`http.req.start::tid1` と`http.req.end::tid1` はスレッド 1 で処理された HTTP リクエストの処理開始と処理終了を示すイベントである．`open::tid1` はスレッド 1 から発生したファイル操作イベントである．このファイル操作イベントはスレッド 1 より発生していること (条件 1) および HTTP リクエストの処理開始時刻と処理終了時刻の間にあること (条件 2) から，`open::tid1` はスレッド 1 で処理された HTTP リクエストに関連するものであって，スレッド 2 で処理された HTTP リクエストに関連するものではない図 4.4 (a) ことがわかる．加えて，`open::tid1` が最初のスレッド 1 で処理された HTTP リクエストに関連付くものであって，2 番目のスレッド 1 で処理された HTTP リクエストに関連するものではない図 4.4 (b) ことも判断できる．

2 つ目の例はプロセスツリーの関係性に基づく関連付けの例である．図 4.5にその概要を示

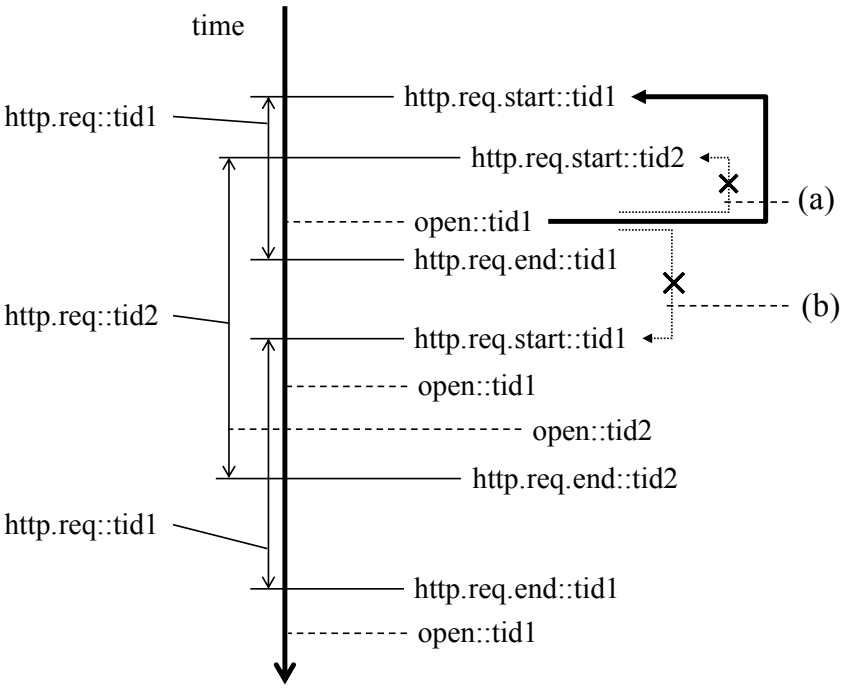

図4.4: TID の関係性に基づく関連付け

す．OS コマンド実行の場合，コマンドは HTTP リクエストを処理するプロセスから生成された子プロセスとして実行される．これは TID の一致に基づく追跡ができないため，PID および PPID の関係性を利用し，プロセスツリーに基づいて関連付けを行う．

図 4.5にプロセスツリーに基づいた関連付けの様子を示す．プロセス 1 が HTTP リクエストを処理する際にコマンド実行が発生し，`fork`システムコールの戻り値より，プロセス 2 が生成されていることがわかる．`execve::pid2` および`open::pid2` はプロセス 2 により発生したイベントである．`fork`システムコールよりプロセス 2 はプロセス 1 から発生していることがわかるため，これら 2 つのイベントはプロセス 1 で処理した HTTP リクエストに関連するものであると判断する．

4.3.2 Alert Correlating フェーズ

従来提案されてきた手法ではどれも異常なシステムコールを検知する手法 [33, 34, 35] や異常な SQL クエリ発行を検知する手法 [36, 37] であるため，アラートとして異常となったシステムコールの情報や SQL クエリの情報を出力することが可能であると考える．そのため，これらの手法がアラートを出力した際はそのアラートを引き起こしたシステムコールの時刻や発行した際の PID/TID，SQL クエリの内容をもとに提案手法で収集したシステムコールや SQL クエリと一意に関連付けが可能と考える．つまり図 4.2の HIDS (a) あるいは (b) いずれ

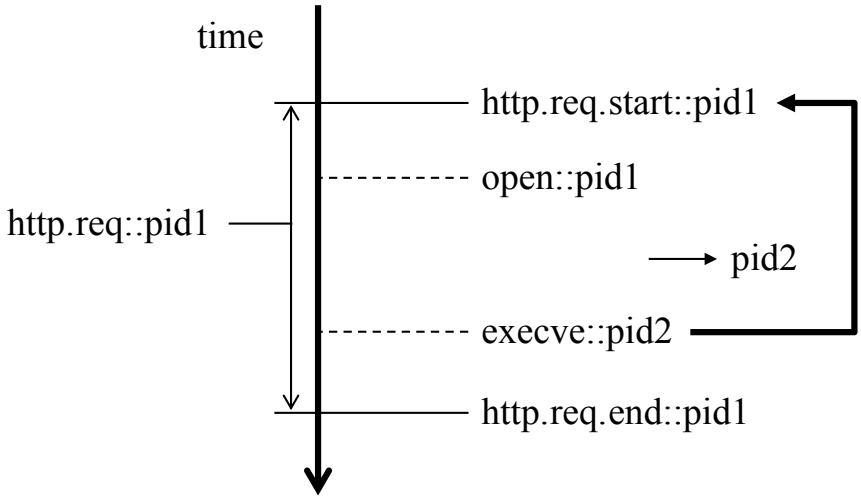

図4.5: プロセスツリーに基づく関連付け

かの形態に該当し，HIDS のアラートからそれを引き起こした HTTP リクエストまで追跡可能と考える．

4.3.3　実装

提案手法を検証するため，プロトタイプ実装を行なった．図 4.6にプロトタイプシステムの全体像を示す．図中の灰色部分が提案手法に関わる部分である．4.3節に示した Collecting フェーズの処理はロギングモジュール (Logging Module) に実装され，4.3.1節および 4.3.2節に示した，Event Correlating フェーズおよび Alert Correlating フェーズの処理は Correlating Module に実装されている．

プロトタイプでは PHP で作成された Web アプリケーションが Apache HTTP Server 上で動作し，DB に MySQL を用いている状況を想定した．Web サーバの動作モデルは Prefork/Worker，どちらも対応できるよう実装を行なった．本実装では Apache HTTP Server と PHP を対象としているが，これは提案手法がこの実装に制限されているという訳ではない．Web サーバが Nginx や lighthttpd 等に変わったとしても同じ要領でイベントを取得可能である．実際，Nginx や lighthttpd でも同様にイベントを取得できることを確認した．

イベントは監視対象のサーバで動作するロギングモジュールによって収集され，収集したイベントは関連付けを行う別サーバに送られる．提案手法では SystemTap [59] を利用してロギングモジュールを作成した．このモジュールによってシステムコールや SQL クエリ発行および HTTP リクエストの収集を行う．SystemTap は，カーネルモジュールを作成しシステムコールやユーザ空間の関数をランタイムフックするためのフレームワークである [60, 61]．ランタイムフックとはプログラム実行時に目的とする関数などが実行された際に追加の処理を行わ

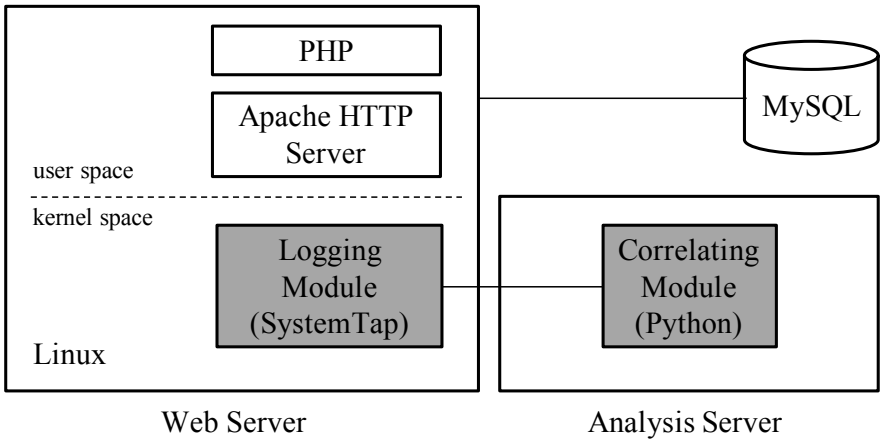

図4.6: プロトタイプシステムの概要

せる仕組みである．SystemTap はユーザが作成したスクリプトを入力として，それをカーネルモジュールとしてコンパイルし，動作中のカーネルに挿入する．SystemTap はパフォーマンス測定に利用された実績 [62] があり，サーバに与える影響は極めて小さいと考えたため本実装に採用した．以下，提案する関連付け手法で必要になる HTTP リクエストの処理開始と処理終了の時刻，システムコール，SQL クエリ発行の取得方法について詳説する．

HTTP リクエストの処理開始時刻と処理終了時刻

Apache HTTP Server の場合，HTTP リクエストは関数`ap_process_request` によって処理されているため，この関数を呼び出す前後でログを出力するようランタイムフックを行なった．HTTP リクエストの処理開始時刻と処理終了時刻は SystemTap が提供する現在時刻をマイクロ秒の精度で取得する関数`gettimeofday_us`を使用することで実現した．HTTP メソッド，リクエスト URL，バージョンやユーザエージェント等の HTTP リクエストに関する情報は関数`ap_process_request` の第 1 引数である構造体`request_rec` から取得した．

システムコール

システムコールの情報は SystemTap の機能を活用することで取得した．SystemTap ではシステムコールの種類のみならず，各システムコールの引数および戻り値まで取得可能である．

SQL クエリ

DB を PHP 言語などのミドルウェアから利用する場合，一般にはその DB が用意するラッパーを経由する．例えば，MySQL の場合クライアントライブラリである`libmysqlclient.so` かネイティブドライバーである`mysqlnd.so` を介して SQL クエリ発行が行われる．そのため，本実装ではラッパーに実装されている SQL クエリ発行を担う関数である`mysql_query`,

mysql_send_query およびphp_mysqlnd_conn_data_query_pub を監視することで，PHP アプリケーションから DB にアクセスする際に使用される SQL クエリを取得した．

4.4　評価

提案手法を関連付け精度，処理速度，使用リソース量の観点から評価を行なった．

4.4.1　関連付け精度

目的　提案手法はイベントの関連付けするための手法であり，正しく関連付けを行えているかが実用性に大きく影響する．そのため，提案手法および既存手法のイベント関連付け精度の評価を行なった．

準備　図 4.7に示す関連付け精度評価用 Web アプリケーションを作成した．URL パラメタに識別子 (uid) となる値を設定して HTTP リクエストを行うと，識別子がシステムコールや DB アクセスのイベントに現れる仕様である．評価の際に利用した DB には追加で定義したスキーマはなく，デフォルト状態のものを利用した．4 行目の fopen 関数によってファイルパス/tmp/$uid にアクセスを行うが，評価時はこのファイルパス対応するファイルを用意しなかった．ファイルアクセスは失敗となるが，ファイルアクセスに相当するシステムコールは発行されるため評価に影響しないと考えたからである．

各イベントに現れる識別子が関連付けられた HTTP リクエストに現れる識別子と一致しており，かつ正しい識別子であるイベントのみが関連付けられている場合，正しい関連付けと判断した．つまり，関連付けるべき識別子のイベントが関連付いていない，あるいは異なる識別子のイベントが HTTP リクエスト関連付けられている場合誤った関連付けと判断した．

評価用 Web アプリケーションに対して同時接続数 1，5，10，50，100 および 500 として 10,000 回の HTTP リクエストを送信し，HTTP リクエストとイベントの取得を行なった．HTTP リクエストの送信には Apache JMeter [63] を利用した．Web サーバには Prefork モードおよび Worker モードの Apache HTTP Server を利用し，サーバパラメタはデフォルト値[*1]を利用した．サーバが Prefork モードの場合，同時接続数が 50 以上およびモードが Worker サーバの場合，同時接続数が 500 以上の場合において，ロギングモジュールの性能限界により取得したイベントの欠損が発生していたため，これらの場合の測定値は除外した．

比較として BackTracker [57] を利用した際の精度評価も行なった．BackTracker の実

[*1] デフォルト値は次の通り．MinSpareServers: 5，MaxSpareServers: 10，MaxClients: 150，MinSpareThreads: 25, MaxSpareThreads: 75, ThreadsPerChild: 25

表4.1: HTTP リクエストとホストイベントの関連付け精度

サーバモード	同時接続数	最大 RPS	平均 RPS	関連付け精度		
				BackTracker		提案手法
Prefork	1	113	66	83.56%	($\Delta = 0.10$)	100%
	5	490	480	88.53%	($\Delta = 0.01$)	100%
	10	521	517	17.65%	($\Delta = 0.01$)	100%
Worker	1	81	48	21.01%	($\Delta = 0.10$)	100%
	5	333	329	12.24%	($\Delta = 0.01$)	100%
	10	346	340	0.23%	($\Delta = 0.01$)	100%
	50	316	312	0.1%	($\Delta = 0.01$)	100%
	100	309	298	0.01%	($\Delta = 0.01$)	100%

装は論文に基づいて著者らが行なった．BackTracker では時間間隔 Δ を指定する必要があるため，今回の評価では Δ を 1s，0.1s，0.01s および 0.001s とした際に最も関連付け精度が高い結果を BackTracker の関連付け精度とした．

結果 サーバモードおよび同時接続数を変化させた場合のイベント関連付け精度を表 4.1に示す．BackTracker は同時接続数が大きくなるほど誤った関連付けを起こしやすくなる傾向にあるが，提案手法は関連付けを誤ることが発生しなかった．今回の実験で秒間最大 521 リクエストまで正しく関連付けが行われることを確認できた．

考察 著者ら管理する個人のブログサイトおよび企業の Web サイトではそれぞれ，最大 RPS (Request Per Second) が 74 および 362 程度であることから，個人のブログサイトや企業の Web サイト程度のアクセスがある環境であれば十分実用可能な性能であると考える．実 Web サイトに対して適用可能かは厳密には最大同時接続数をもとに評価する必要があるが，Web サーバのアクセスログから同時接続数を把握することは困難であるため，本論文では最大 RPS をもとに適用可否の評価を行なった．数千数万のアクセスが同時発生する大規模環境については後述するログ量の問題やロギングモジュールの性能面の問題があるため，提案手法およびそのプロトタイプは企業や個人等の一般的な Web サーバへの適用を想定している．

4.4.2 処理性能

提案手法では Web サーバ内でログを取得するため，Web サーバのパフォーマンスに影響を与えることが想定できる．パフォーマンス低下が実用的である範囲内であるかを評価するた

```
1  <?php
2  $uid = $_GET['uid'];
3  // file access
4  $fp = fopen("/tmp/{$uid}", 'r');
5  fclose($fp);
6  // network access
7  $socket = socket_create(AF_INET, SOCK_STREAM,
8                          SOL_TCP);
9  $result = socket_connect($socket, '127.0.0.1',
10                          $uid);
11 socket_close($socket);
12 // command
13 system("cat /tmp/{$uid}");
14 // db access
15 $mysqli = new mysqli('127.0.0.1',
16                       'root', 'root',
17                       'information_schema');
18 $sql = "SELECT {uid} FROM information_schema";
19 $mysqli->query($sql)->close();
20 ?>
```

図4.7: イベント関連付け精度評価用 Web アプリケーション

め，表 4.2に示す著名な Web アプリケーションが動作する Web サーバに対して本手法を適用した際のスループット低下率，最大メモリ使用量，最大 CPU 使用率，平均ディスク使用量を調べた．

Web サーバの処理性能に影響を与えるのはイベントロギングを行うカーネルモジュールであり，関連付けを行う部分は Web サーバの処理性能に対して影響を与えないため，この部分の性能測定は実施しなかった．性能測定に使用したマシンのスペックは Intel Xeon 2.40-GHz CPU，8 GB RAM，128 GB HDD，CentOS 7.1 であった．

HTTP リクエストのスループット

目的 HTTP リクエストのスループット低下は Web ページの表示速度の低下を意味し，ユーザビリティを損なうことになってしまうため，モジュールやシステムの導入においては HTTP リクエストのスループット低下を抑えることが必要である．そのため，提案手法のロギングモジュールを導入した際の HTTP リクエストのスループット低下を評価した．

表4.2: 性能測定に用いた Web アプリケーションの一覧

名称	バージョン	利用用途
Joomla	3.4.4	コンテンツ管理
Drupal	7.3.1	コンテンツ管理
MediaWiki	1.26.2	コンテンツ管理
WordPress	4.4.1	コンテンツ管理

準備 Web アプリケーションのパフォーマンス測定ツールとして有名な Apache JMeter を利用して測定を行なった．スループット値は 1,000 リクエストを同時接続数 10 の状態で送信し計測することを 1 セットとし，5 セット測定した際の平均で求めた．Web サーバには Prefork モードの Apache HTTP Server を利用し，サーバパラメタはデフォルト値を利用した．T_d と T_e をそれぞれイベントロギングを無効にした場合と有効にした場合のスループットとし，スループット低下率は $\frac{T_d - T_e}{T_d}$ で求めた．

結果 表 4.3に測定結果を示す．どのアプリケーションも平均スループット低下率が 5% 以下であった．Joomla の平均スループット低下率が最も大きい結果となった．

考察 イベント収集を無効にした際の Joomla の RPS が 13.56 であるため，4.62% のスループットの低下はリクエストあたり約 73 ミリ秒の遅延を意味する．Joomla では各機能のアクセスに対して約 5 回のリクエストを行うことから，ユーザが体感する遅延は 0.37 秒程度である．1 秒程度の遅延が発生するのであればユーザビリティは大きく損なわれないとの調査 [64] があるため，スループット面では実用的であると考える．

スループット低下には 2 つの要因が存在する．1 つ目の要因はユーザ空間のプログラムに対するランタイムフックがある．提案システムではカーネルモジュールとして動作するため，システムコールはカーネル空間から取得可能であるが，HTTP リクエストや SQL クエリ発行に関する情報はユーザ空間のプログラムにフックを行うことが必要である．カーネル空間からユーザ空間に対してランタイムフックを行う際は uprobe [65] という仕組みが利用されるが，uprobe を介することでスループット低下率は更に増加する．そのため，uprobe を介して取得する必要があるイベントの割合がスループット低下の要因につながっていると考える．表 4.3中の UP 割合は全てのイベントの内，uprobe を介するイベントの割合である．表 4.3中の UP 割合およびログ量はリクエストあたりの値になっている．UP 割合では Joomla の場合およびログ量では MediaWiki の場合を除いて，概ねスループット低下率とは比例した関係であると考える．2 つ目の要因は出力するログ量である．ロギングモジュールが出力する情報が多いほど出力に時間を要するためスループット低下も大きいと考える．そのため，この 2 つの要因を削減

表4.3: スループットの低下率

名称	機能	RPS	平均	低下率	平均	UP 割合	平均	ログ量	平均
Joomla	view top	13.01	13.56	4.79%	4.62%	16.86%	22.21%	35.50KB	51.126KB
	login	19.24		3.84%		16.18%		30.83KB	
	create post	9.09		5.47%		39.09%		60.63KB	
	view post	12.69		4.39%		16.70%		77.53KB	
Drupal	view top	18.47	15.84	2.69%	2.67%	43.83%	49.13%	36.79KB	45.06KB
	login	16.80		4.44%		53.69%		61.02KB	
	create post	14.23		0.17%		46.67%		42.50KB	
	view post	18.43		2.78%		45.03%		31.59KB	
	comment	11.27		3.28%		50.84%		53.39KB	
MediaWiki	view top	8.95	6.89	2.97%	2.49%	32.13%	37.02%	45.32KB	62.67KB
	login	6.03		1.98%		29.00%		60.77KB	
	create post	5.68		2.77%		41.92%		81.91KB	
WordPress	view top	10.02	9.50	1.81%	1.54%	5.72%	4.89%	45.46KB	40.91KB
	login	14.73		0.62%		3.29%		34.67KB	
	create post	3.55		1.52%		4.97%		47.17KB	
	view post	12.55		2.08%		3.86%		42.32KB	
	comment	5.74		1.70%		6.61%		34.96KB	

することで今後よりスループット低下を抑えることが可能性が高いと考える．

リソース利用量

目的　Web サーバはそのアクセス数に応じて利用できるリソースが限られている場合もある．そのため，提案手法のロギングモジュールを利用した際の最大メモリ使用量，最大 CPU 使用率，平均ディスク使用量を調べ，過剰なリソース利用がないかを評価した．

準備　ディスク使用量は出力されたログファイルを `gzip` で圧縮した際の値を計測した．圧縮した値を用いて比較する理由は，プロトタイプ実装のログフォーマットに冗長な部分があり，ログフォーマットの効率化によって改良可能なため，この部分による影響を除外するためである．

結果　リソース利用量の計測結果を表 4.4に示す．

考察　50MB 程度のメモリ使用量および 4% 程度の CPU 使用率は近年の Web サーバに問題ない使用量だと考える．しかし，取得するログによるディスク使用量は一般的な場合に比べると非常に多い．法令やガイドラインでは，様々なセキュリティインシデントへの対処のためにログの保存期間として 3 か月間や 1 年間，あるいはさらに長期間を定めている [66, 67]．一般的な Web サーバの圧縮時のアクセスログのディスク使用量はリクエストあたり約 20B であり，本手法の場合のログのディスク使用量はリクエストあたり平均約 8.75KB であるため，通常のアクセスログに比べ，400 倍を超える容量となるが，提案手法で取得するログは，通常のアクセスログの補完として，攻撃発生時に即時確認用として短期保存を想定することで運用可能と考えている．

表4.4: リソース使用量

名称	メモリ	CPU	ディスク/リクエスト
Joomla	53.4MB	7.80%	12.0KB
Drupal	53.4MB	9.40%	11.3KB
MediaWiki	53.3MB	4.60%	10.8KB
WordPress	53.3MB	4.00%	7.00KB

4.4.3 事例分析

提案手法の効果は HIDS アラートが発生した際にどの HTTP リクエストがそのアラートを引き起こしているかを特定できる所にあり，人手で特定する必要がないため，アラートに対する調査の時間短縮に貢献できる所にある．提案するイベント関連付け手法を活用した際の効果について 2 つの事例をもとにどのように HIDS アラート調査の時間短縮に貢献したかを評価した．評価に際しては Mutz らの手法 [35] を参考に著者らが実装した HIDS を活用した．

事例 1

実験環境にて脆弱性が存在する Web アプリケーション Drupal 8.5.0 を用意し，OS コマンドインジェクションの脆弱性 (CVE-2018-7600) に対して攻撃を行なった．HIDS はコマンドの頻度から異常なコマンド実行を検知することが可能である．表 4.5に提案手法で出力された単一のリクエストの処理の開始から終了までのイベントの一部を抽出した結果を示す．HIDS では No.6, 17 および 18 のイベントを異常として検知していた．提案手法にて関連付けた No.1 の HTTP リクエストの情報によりこの攻撃は URL パス`/user/register` に対するものであることがわかる．この情報により，管理者は脆弱性に対するパッチが提供される前に，この URL に対するアクセス制限を設けるなどの暫定対処を行うことが可能となる．

人手でこれらのイベントを関連付ける場合，まず，No.6, 17 および 18 の HIDS が異常判定したイベントから No.1 に示す HTTP リクエストを探す必要がある．イベントの発生時刻が近いことに基づき関連付ける必要がある．イベントの時刻精度が十分でない場合，あるいは短時間に複数リクエストが発生している場合，正しく関連付けるには Web アプリケーションの仕様や Web サーバの状況を鑑みて行う必要があり，時間を要すると考える．また，攻撃となった HTTP リクエストによって何がなされたのかの全体像を把握するためには，SQL クエリの発行ログやシステムコールログ等から No.2 から No.5 および No.7 から No.16 のイベントを関連づける必要がある．通常，SQL クエリ発行ログにはどのプロセス／スレッドで発生したものなのかを示す識別子は存在しないため，Web アプリケーションの仕様に基づいて関連

表4.5: Drupal に対する OS コマンドインジェクションに対する関連付け結果

No.	HIDS 異常判定	種類	内容
1		http	POST /user/register?element_parents=account/mail/%23value
2		syscall	open, .htaccess, O_RDONLY:O_CLOEXEC
3		syscall	net, connect, /var/lib/mysql/mysql.sock
4		db	SET NAMES utf8mb4
5		db	SELECT cid, data, ... FROM cache_container WHERE cid IN (*) ORDER BY cid
6	✓	syscall	execve, sed -ie s/allow,deny/deny,allow/g .htaccess
7		syscall	open, /lib64/libc.so, O_RDONLY:O_CLOEXEC
8		syscall	open, /proc/meminfo, O_RDONLY:O_CLOEXEC
9		syscall	connect, /var/run/nscd/socket
10		syscall	open, /etc/nsswitch.conf, O_RDONLY:O_CLOEXEC
11		syscall	open, /etc/ld.so.cache, O_RDONLY:O_CLOEXEC
12		syscall	open, /etc/passwd, O_RDONLY:O_CLOEXEC
13		syscall	open, /lib64/libpcre.so, O_RDONLY:O_CLOEXEC
14		syscall	open, /proc/filesystems, O_RDONLY
15		syscall	open, /usr/lib64/charset.alias, O_RDONLY,O_NOFOLLOW
16		syscall	open, .htaccess, O_RDONLY
17	✓	syscall	open, ./sed3wZccC, O_RDWR,O_CREAT,O_EXCL
18	✓	syscall	rename, ./sed3wZccC, .htaccess

づける必要があり，時間を要すると考える．提案手法により上記の関連付け作業が必要なくなるため，分析による時間を短縮できると効果があると考える．

事例 2

我々は HIDS およびブログサイトを日々運用しており，2019 年 4 月 1 日から 2019 年 7 月 22 日の間で 282,585 件の HTTP リクエストを観測し，その内 49 件の HTTP リクエストによるイベントが異常として検知されていた．検知したイベント全てを確認し，侵害を示すものではなかったことから 49 件の検知は全て誤検知であった．

表 4.6に誤検知した際に提案手法が関連付けたイベントの一部を示す．HIDS は No.2 から No.11 まで全てのイベントを異常として検知していた．しかし，関連付いた HTTP リクエストの内容を確認しても攻撃コードが含まれていなかった．今回誤検知した動作は我々が運用している Web アプリケーション WordPress の更新確認に伴うものであるため，HIDS の誤検知であると確認できた．このように HTTP リクエスト情報が関連付いていることにより，HTTP リクエストに攻撃コードが含まれているか否かの情報を加味して攻撃に対応できるようになるため，管理者にとってより容易に攻撃に対する対応が可能となる．No.4 のイベントは攻撃によってもよく引き起こされるため，HIDS の結果だけから判断すると侵害されていると判断していまう可能性も高いが，HTTP リクエストの情報とともに判断することによってより早期に誤検知と判断できる．

表4.6: 誤検知に対する関連付け結果

No.	HIDS 異常判定	種類	内容
1		http	GET /category/events
2	✓	syscall	open, /lib64/libsoftokn3.so, O_RDONLY:O_CLOEXEC
3	✓	syscall	open, /lib64/libfreeblpriv3.so, O_RDONLY:O_CLOEXEC
4	✓	syscall	open, /etc/passwd, O_RDONLY
5	✓	syscall	open, /etc/pki/nssdb/pkcs11.txt, O_RDONLY
6	✓	syscall	open, /proc/sys/crypto/fips_enabled, O_RDONLY
7	✓	syscall	open, /lib64/libnsssysinit.so, O_RDONLY:O_CLOEXEC
8	✓	syscall	open, /etc/pki/nssdb/cert9.db, O_RDWR:O_CREAT:O_CLOEXEC
9	✓	syscall	open, /etc/pki/nssdb/key4.db, O_RDWR:O_CREAT:O_CLOEXEC
10	✓	syscall	open, /etc/pki/nss-legacy/nss-rhel7.config, O_RDONLY
11	✓	syscall	open, /lib64/libnsspem.so, O_RDONLY:O_CLOEXEC

4.4.4 提案手法の実用時における考慮事項

HIDS アラートの情報量による制限

4.3.2節で述べたように提案手法では HIDS アラートがイベントと関連付けられることが前提条件となる．一般に利用されている HIDS の中にはアラートに出力できる情報が限られている場合がある．この場合，HIDS がアラートを出力しても，提案手法で取得したイベントと関連付けることができない．例えば Tripwire Enterprise [68] を始めとする HIDS 製品では，アラートとホストイベントを関連付けられる情報がアラート中に存在しないため，本手法は適用できない．

サーバ改ざんの可能性

攻撃者が Web サーバの制御を取得できてしまった場合に，提案手法を無効化できる恐れがある．提案システムではカーネルモジュールを利用してイベント取得を行なっている．そのため，攻撃者が管理者権限を奪取でき，またこのカーネルモジュールを特定した場合は，カーネルモジュールを停止することで提案手法は利用できなくなってしまう．この制約に対しては完全性のチェックを実行時に行う仕組みを導入することで今後，解決可能であると考える．

4.5 関連研究

NIDS はパケットや通信データから特徴量を抽出し，攻撃検知を行う．WAF は NIDS の領域において，Web アプリケーションに特化したものである．研究は古くから行われている [1, 32, 31]．NIDS はシステム改変が少ないことから導入が容易であるが，失敗した攻撃もアラートとして通知してしまうため，管理者の調査すべきアラートが大量になってしまい，管理者の負担が大きくなりがちである．

HIDS はホストであるサーバや端末の OS から収集できる情報を利用して攻撃検知を行う．

サーバ改変を要するため，導入時にアプリケーションの動作に悪影響がないか検証を再度行う必要があるため手間を要する．システムコールの順序やシステムコールの引数を特徴量として異常検知を行う手法 [33, 34, 35] や，Web アプリケーションに特化し，DB アクセスの際に利用される SQL クエリを特徴量として異常検知を行う手法などが存在する [36, 37]．これらの手法では OS や DB など被害が発生する部分から特徴量を抽出するため，攻撃の成否を判定でき，攻撃成功時のみを捉えてアラートを通知することができると考えれる．NIDS を比較した場合，HIDS は失敗した攻撃に対しては検知せず，攻撃成功時のみを検知するため，NIDS より精度の高い通知が可能と考える．しかしながら，HIDS には HTTP リクエストの情報が関連付いていないため，被害が発生させた原因となった Web アプリケーションや攻撃元といった被害の再発防止のための暫定対処に寄与する情報は管理者がサーバのログから調査する必要がある．

NIDS の課題を鑑み，著者らはネットワーク通信におけるリクエストおよびレスポンスを両方監視し，リクエストに含まれる攻撃コードをエミュレーションし得られる攻撃の痕跡がレスポンスに含まれる場合に攻撃の成否を判定する手法 [69] を提案した．これより，一部の攻撃については攻撃が成功したという結果をもとに優先的に調査できるようになった．この手法ではネットワーク通信に攻撃の痕跡が現れる攻撃のみ判定可能であり，サーバ内のファイルの内容を改変するあるいは DB の内容を消去するといったホスト内部に閉じた痕跡を残す攻撃に対しては成否を判定できない．

4.6　結語

NIDS は導入しやすいが，失敗した攻撃に対してもアラート通知を行うため，管理者が対応すべきアラート通知が大量になってしまう．HIDS はホストで発生したイベントを利用するため，アラートが攻撃が成功した場合となる可能性が高く，より精度の高い通知が可能である．しかし，被害の原因調査に必要な攻撃対象の Web アプリケーションの URL や攻撃元など，攻撃となった HTTP リクエストに関する情報を出力できないため，人手の調査を要し時間がかかってしまう．本研究では，HIDS の入力であるシステムコールや SQL クエリ発行をそれらを発生させた HTTP リクエストを，処理したスレッドの ID と高精度な処理開始および終了時刻に基づいて HTTP リクエストと関連付けることで，HIDS で検知した際に管理者が攻撃対象の Web アプリケーションに関する情報を瞬時に特定できるようにした．評価では，提案手法が誤った関連付けをすることがなく，Web アプリケーションに与えるパフォーマンス低下を 5% 以下に抑え，実用的であることを示した．今後の課題として，容量を抑えたログの記録方法の検討や，高いスループットが要求される環境においても提案手法を利用できるようパフォーマンス低下を更に抑えたロギングモジュールの作成に取り組む．

第 5 章

攻撃コードのエミュレーションに基づく攻撃の成否判定手法

5.1 緒言

4章で述べたように，HIDS の仕組みを活用することで被害があった際，つまり攻撃が成功した場合を判定できるようになるため，これらの HIDS のアラートをより優先的に対応することで，より効率的な対応が可能となる．しかし，脆弱性の確認を意図とした攻撃のように攻撃の痕跡がサーバ内に残らない場合には攻撃の成否を判定できない．また，HIDS は Web サーバを改変して導入するため，商用環境では動作保証を行うことが難しかったり，導入によりパフォーマンスの低下が必ず発生したり，導入しずらい側面も存在する．

本研究ではネットワーク通信から攻撃の成否を判定することで，アラートの重要度を決定する．つまり，攻撃が成功しているのであれば，そのアラートはより重要と判定することでセキュリティオペレーションの効率化につなげる．関連研究ではサーバ内のシステムコールや命令列を観測したり [33, 70, 71, 72, 73], 攻撃の複数の状態をルールとして定義したり [74, 75, 76], することで成否を判定している．しかし，このような仕組みの導入ではシステム改変が必要となるため，商用環境において導入障壁が大きかったり，攻撃の状態を定義するルール作成を行うため利用者に要求される知識レベルが高いという制約が存在する．本研究ではシステム改変不要および比較的低い知識レベルの利用者でも運用可能という 2 つの要件を満たす手法を提案する．

本研究の貢献は以下の通りである．

- 本研究ではシステム改変が不要かつ，比較的低い知識レベルの利用者でも運用可能な成否判定手法を提案した．提案手法では攻撃コードのエミュレーションを行い，攻撃の痕跡である IOC (Indicator Of Compromise) を抽出する．その後，IOC が HTTP レス

ポンスに含まれるか否かで攻撃の成否を判定し，アラートの重要度を決定する．

- 人工的に作成したデータと実環境のデータで評価を行い，それぞれ，62.9% および 52.1% の不要なアラートを削減できること，WAF のアラートでは気づけなかった成功した攻撃を発見できること，約 98% のリクエストを 1 秒以内に処理できることを示し，提案手法の効果を示した．約 44% のアラートが見過ごされている状況を鑑みると目標を達成できたと考える．

5.2　攻撃の成否

サーバに対する攻撃の結果は，成功と失敗に大別される．攻撃の成功とは，攻撃者の意図通りに攻撃が行われている状態のことである．例えば，以下のような Web サーバに対する任意の OS コマンド実行を狙う攻撃があるとする．`GET /index.php?path=;cat%20/etc/passwd` 攻撃者が挿入したコマンド`cat /etc/passwd`が実行され，ファイル`/etc/passwd` の内容が以下のようにレスポンスに表示されていた場合攻撃は成功していると判断される．

```
<html><h1>Welcome!</h1>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin...</html>
```

しかし，以下のようにファイル`/etc/passwd` が表示されていない場合，攻撃者が挿入した OS コマンドは実行されていないため，攻撃は失敗していると判断できる．

```
<html><p>Page not found.</p>...</html>
```

5.3　提案手法

提案手法の全体像を図 5.1に示す．提案手法では 5 つのステップにより攻撃成否の判定を行う．

提案手法は，WAF 等で攻撃検知された Web 攻撃の HTTP リクエストおよび HTTP レスポンスのデータを入力とする．出力の判定結果は攻撃成功，攻撃失敗および判定不可の 3 つである．まず，攻撃検知された HTTP リクエストに対して，攻撃カテゴリの判定および攻撃コードの抽出を行う (**Category Identification**)．次に，抽出した攻撃コードから複数の部分的な攻撃コード (攻撃コード候補) を生成し (**Candidate Generation**)，生成した複数の攻撃コード候補をエミュレータで実行する (**Code Emulation**)．次に，エミュレータの出力を整形し，攻撃の痕跡 (IOC) を抽出する (**Indicator Extraction**)．最後に，IOC を HTTP レスポンスと比較し，攻撃が成功したか否かを判定する (**Attack Verification**)．以降，各節にて各処理ステップを詳しく説明する．

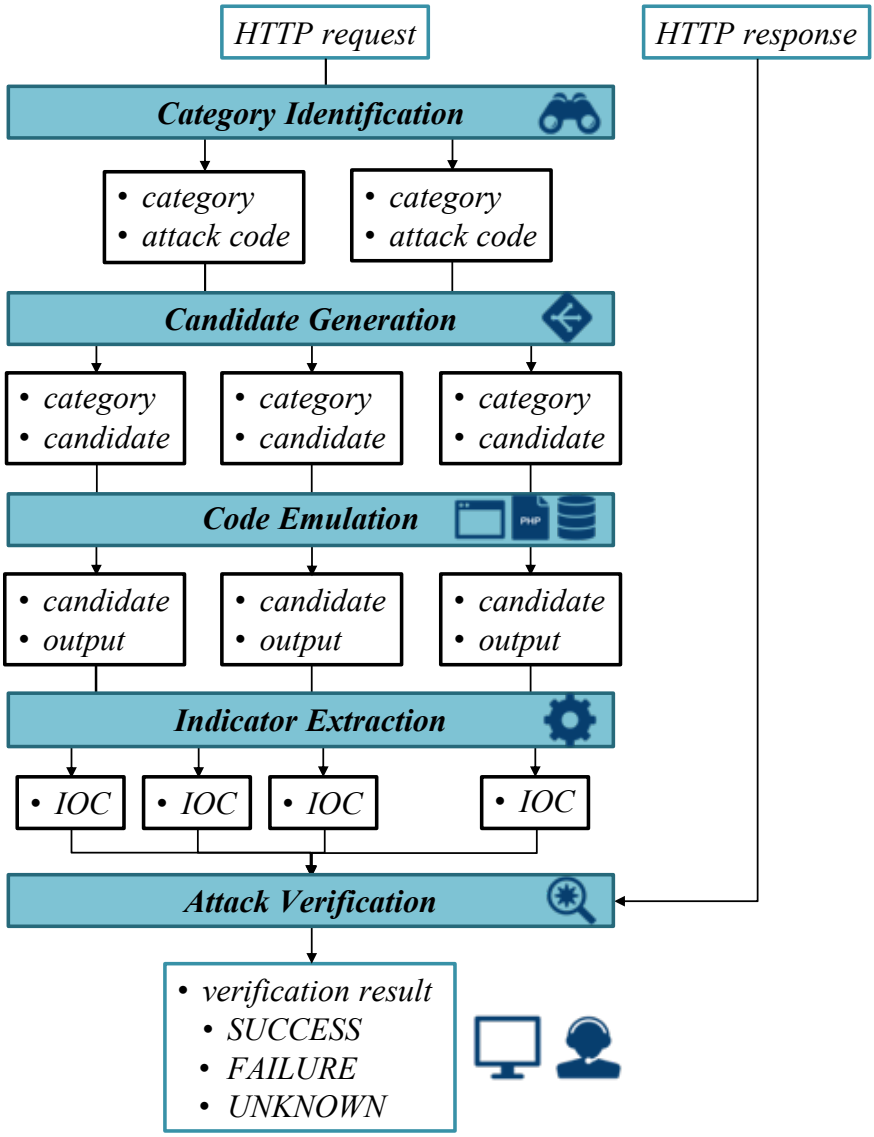

図5.1: 提案手法の各処理とデータの流れ

5.3.1 Category Identification

攻撃において脆弱性を悪用する部分はアプリケーション毎に異なり多様であるが，脆弱性を悪用した際に行う動作は限定的である．本研究ではこの動作を表 5.1に示す 4 つのカテゴリに分ける．この 4 つのカテゴリは OWASP Top 10[*1]でも示されるように脆弱性を悪用する攻撃の代表的なものである．

[*1] 国際的な Web アプリケーションのセキュリティコミュニティである OWASP が公開している対策すべき重要な Web アプリケーションの脆弱性

攻撃カテゴリを判別する理由は 2 つある．1 つ目はエミュレーションの正確性を高めるためである．正しくないエミュレータを利用して得られた攻撃の痕跡はその攻撃の成否を断定する根拠として乏しいので，攻撃の実行に適切な環境を選択することは精度向上のために重要である．例えば，OS コマンド実行である攻撃コード`cat /etc/passwd`を XSS (Cross-Site Scripting) と誤判定した場合，この攻撃コードの文字列が出現しているか否かで攻撃の成否を判定するになってしまうため誤判定を誘発する．2 つ目は **Code Emulation** ステップの処理時間を短縮するためである．Web アプリケーションを構成する環境は OS の種類，実行環境の種類，DB の種類が一様ではない，また **Candidate Generation** ステップにより複数の攻撃コードの候補が生成されるため，複数回の攻撃コードのエミュレーションを Web アプリケーションを構成する全ての環境で実施すると処理に時間がかかってしまう．そのため，攻撃カテゴリを判別し，攻撃コードに対して適切な環境のみでエミュレーションを行うことで処理時間を短縮する．

攻撃カテゴリの判断には各カテゴリの固有表現に攻撃コードが一致するかどうかで決定する．RCE (Remote Command/Code Execution) カテゴリの攻撃の判定には，OS コマンドやプログラミング言語固有の表現が攻撃コード中に存在するかどうかによって判断する．また，表 5.2に示すように，RCE カテゴリには OS コマンドあるいはプログラミング言語の差異を区別するためのサブカテゴリを設ける．OS コマンドの固有表現には，`/bin`や`/usr/bin`以下のプログラム名および，`bash`等のビルドインコマンドの一覧を利用する．PHP 等のプログラミング言語の固有表現には関数名の一覧等を利用する．SQLI (SQL Injection) カテゴリの固有表現には DB にアクセスする際に使用する SQL 句を利用する．PT (Path Traversal) カテゴリの固有表現にはファイルパスの相対参照をする際に必要な`../`といった表現，および Web アプリケーションとは直接関連しないパス (`/etc`，`/proc`) ，バックアップファイルパス (`*.bak`, `*~`) 等の表現を利用する．XSS の固有表現には HTML タグや JavaScript の関数名を利用する．表 5.2に各カテゴリに判別するための固有表現の一例を示す．固有表現の設定は利用者で調整を行うことも可能だが，入手容易な OS コマンドの一覧や関数の一覧等を設定するのみであり，入手困難な攻撃方法に関する情報ではないため，利用者に要求する負担は小さいと考える．

判定では，攻撃コードに対する URL エンコード等は全てデコードされている状態を前提としている．また，1 つの攻撃コードが複数のカテゴリに分類される場合もある．例えば，以下の攻撃コードは`cat`の部分により RCE カテゴリと判定され，`../`の部分で PT カテゴリと判定される．

```
cat ../../etc/passwd
```

攻撃カテゴリを判定できない場合は判定不可の結果を出力し，処理を終える．

表5.1: 攻撃カテゴリ一覧

カテゴリ	説明
RCE	OS コマンドやプログラムコードを悪用する攻撃
SQLI	SQL コマンド (DB の機能) を悪用する攻撃
PT	ファイル操作を悪用する攻撃
XSS	HTML や JavaScript を悪用する攻撃

表5.2: 攻撃カテゴリを判別するための固有表現の例

カテゴリ	サブカテゴリ	固有表現例

RCE	SH
	PHP
	...
		`echo, cat, uname`
		`phpinfo(), $_GET, $_POST`
		...

SQLI		`SELECT, UPDATE`
PT		`../, /etc/, /proc/, *.bak, *~`
XSS		`<script>, <iframe>, alert(`

5.3.2 Candidate Generation

攻撃コードによってはそのままではエミュレーションすることができない場合が存在する．例えば，SQL インジェクション攻撃では，任意のステートメントを実行するのに

```
UNION SELECT version()#
```

のようにUNIONを付与したコードの実行を試みる．しかし，このステートメントをそのまま実行すると，UNIONという SQL 句が余分となり，正しく実行できない．このステップではエミュレーションの正確性を高めるために，実行する攻撃コードを整形した候補を複数生成する．RCE カテゴリの攻撃コードに対してはコマンド名を攻撃コードの先頭とした候補を抽出する．SQL カテゴリの攻撃コードに対しては (SELECT, UPDATE) などの SQL コマンドを攻撃コードの先頭とした候補を抽出する．XSS カテゴリの場合，HTML タグ単位で候補を生成する．なお，元々の攻撃コードも候補の 1 つとして扱う．上記の例の場合，

```
UNION SELECT version()#
SELECT version()#
```

の 2 つの攻撃コード候補が生成される．

5.3.3 Code Emulation

攻撃カテゴリに応じて攻撃コードを実行するエミュレータを用意し，攻撃コードをその環境にて実行する．エミュレーションが終了しないこと (例えば`sleep`コマンドの実行) を避けるため，エミュレーションにはタイムアウトを設ける．

エミュレータでは攻撃カテゴリ毎に異なる手法で攻撃コードを実行する．RCE カテゴリの攻撃に対しては OS コマンドを実行できる GNU bash や，PHP コードを実行できる PHP インタプリタ等を用いて攻撃コードを実行する．SQLI カテゴリの攻撃に対してはインストールが終了した初期状態の DB サーバを用意し，SQL 文を実行する．PT カテゴリの攻撃に対しては攻撃がアクセスしようとしているファイルパスあるいはファイル名を OS のディレクトリおよびユーザが指定した Web アプリケーションの設定ファイルが保管されているディレクトリから検索し，存在すればそのファイルを表示することでエミュレーションを行う．バックアップファイルパス (`*.bak`, `*~`等) の場合は`*`の部分を抽出し，そのファイルパスが存在すれば，ファイルを表示することこでエミュレーションを行う．XSS カテゴリの攻撃は攻撃コードそのものが HTTP レスポンスに現れるため，エミュレーションは行わず，HTTP レスポンス中での攻撃コード自体の有無によって成否判定を行っている．XSS カテゴリの成否判定については既存手法 [77] などを用いても構わない．

幅広い攻撃に対応するために，エミュレータ環境には攻撃コードを実行する環境である OS やミドルウェアを複数用意する必要がある．また，エミュレータ環境自体が攻撃コードによって改変される可能性が高いため，以下の 3 つの条件が必要となる．

1. 複数の OS やミドルウェアで利用可能であること
2. プログラムを実行した後でも実行前の状態に戻せること
3. 実行前の状態を高速に復元できること

QEMU [78] 等の OS 仮想化技術ではなく，より軽量に実行・復元が可能である Linux コンテナ技術である Docker [79] を活用して実装を行うことで上記の 3 つの条件を解決した．

実際の環境とエミュレーション環境の差異が生じることで，成否判定の結果に悪影響を及ぼすことがあるため，広く利用される OS や実行環境を用意しエミュレーションを行う．例えば SQLI カテゴリの攻撃コードに対しては MySQL と PostgreSQL 両方の環境でエミュレーションを行う．提案手法では脆弱性を悪用する部分をエミュレーションするのではなく，脆弱性を悪用した後に実行される攻撃コードのエミュレーションを行う．そのため，多種多様な Web アプリケーションの環境を用意する必要はなく，限られた OS，実行環境や DB を用意すればよい．

エミュレーションする際の特徴は Web アプリケーションの脆弱性ではなく，攻撃者が実行

したいコードあるいはコマンドに着目していることである．そのため，多種多様な Web アプリケーションをエミュレートする必要はなく，Web サーバで利用する OS やミドルウェアといった限定的な環境を用意できればよい．

5.3.4 Indicator Extraction

エミュレータで攻撃コードを実行すると結果が標準出力あるいは標準エラー出力に出力される．そのため，これらの出力を利用して攻撃が成功したことを示す指標である IOC を抽出する．

IOC 候補の抽出

標準出力に出力された全ての内容を HTTP レスポンスとそのまま比較すると，検知漏れが発生する恐れがある．これは Web アプリケーションによっては画面表示が一部分に制限されていたり，空白等が変換されて，一連の出力にならない等の仕様があるからである．標準出力に対する以下に示す IOC 候補抽出処理を行う．なお，IOC 候補の集合の初期状態は空の集合である．

1. 出力が複数行の場合は，各行をそれぞれの IOC 候補とする
2. 出力の各行がタブ文字もしくは空白文字で区切られている場合は，タブ文字で区切った文字列をそれぞれ IOC 候補とする

不適切な IOC 候補の除外

IOC は攻撃が成功したことを示す痕跡であるため，攻撃失敗時のレスポンスに現れると誤検知となってしまう．例えば，攻撃コードが`echo 123`，`echo welcome`の場合，抽出される IOC は`123`，`welcome`となるが，Web ページとしては頻出する表現であるので，攻撃が失敗していても成功したと誤検知してしまう．このような誤検知を多発させないために，不適切な IOC 候補を除外する処理を行う．以下に除外条件を示す．

1. IOC の文字列長が T 以下 (T は予め指定する)
2. 汎用的な HTML タグ (例：`<html>`，`<head>`，`<body>`等) が存在
3. 一般的に利用される英単語等の文字列 (例：`welcome`，`login`等) が存在

IOC 候補の正規化

最後に，実際の環境とエミュレーション環境の差異を少なくするため，IOC 候補に対して正規化処理を行う．正規化処理では変化しやすい部分である日付や時刻などの数字等

を正規表現に変換する．これらの表現は予め用意する．例えば，lastコマンドを実行時にはwtmp begins 15:55:59 2017という文字列が標準出力として得られるが，本処理によってwtmp begins \d+:\d+:\d+ \d+のような IOC 候補に変換される．正規化できない場合は正規化を施さずに処理を終える．

残った IOC 候補を攻撃の成否判定に利用する IOC とする．残った IOC 候補がない場合，攻撃の成否を判断することができないことから，判定不可の結果を出力し，一連の処理を終える．

5.3.5 Attack Verification

このステップでは HTTP レスポンスに IOC が含まれているか否かで攻撃の成否を判定する．前ステップで IOC が抽出されなかった場合は成否を判定できないため判定不可となる．IOC が存在し，HTTP レスポンスに IOC が含まれる場合は攻撃コードが実行されたとみなせるため，攻撃が成功したと判定する．そうでない場合，攻撃が失敗したと判定する．

5.4 実装

提案手法を実装したシステムの全体像を図 5.2に示す．HTTP リクエストおよびレスポンスの取得には OSS のネットワーク監視ツールである Zeek (Bro) [80] を利用した．攻撃カテゴリの判定および攻撃成否の判定は Python で実装した．攻撃コードをエミュレートする環境には Docker を採用した．Docker は Linux コンテナ技術を活用しており，OS 仮想化に比べ起動・復元が高速に行える．

提案システムでは攻撃が発生する度，エミュレータとなる Docker Container を作成し，その環境内で攻撃コードを実行する．なお，エミュレーション時に外部への攻撃を防ぐため，ネットワーク通信は全て遮断した．この設定による影響については 5.6節にて述べる．PT カテゴリの攻撃に対しては OS のファイルをエミュレーションに活用する以外にも，Web アプリケーション固有のファイルを追加することで，判定可能な攻撃を増やすことも可能である．実験では，頻繁に発生するコンテンツ管理システム (CMS) である WordPress，Drupal および Joomla などの設定ファイルを追加した．一例としては WordPress の設定ファイルであるwp-config.phpなどが存在する．

5.5 評価

判定精度および処理性能の観点から評価を行った．利用したデータセットの概要を表 5.3に示す．評価には仮想環境で人工的に作成したデータセット D_{lab} および大規模実ネットワーク環境のトラフィックから生成したデータセット D_{real} の 2 種類のデータを用いた．精度評価に

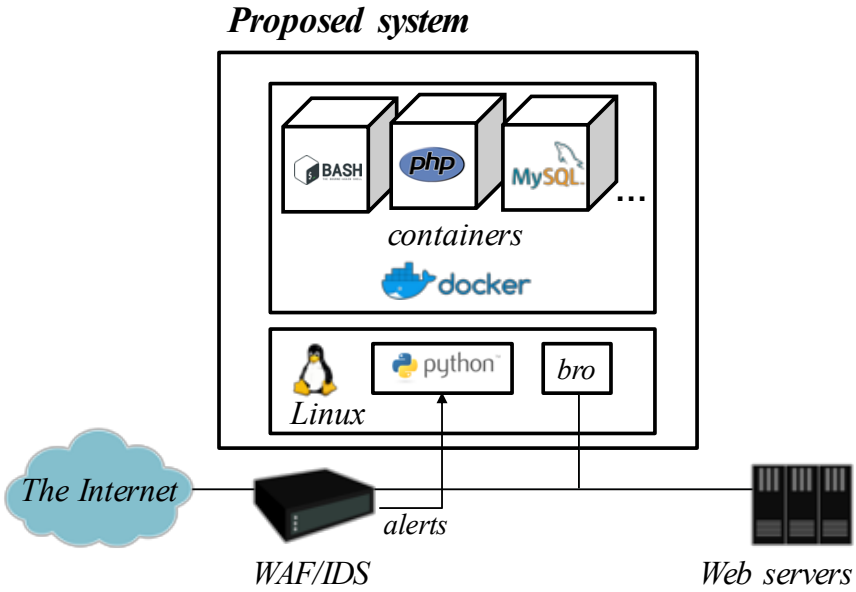

図5.2: システム構成

表5.3: データセット概要

データセット	D_{lab}	D_{real}
観測環境	仮想環境	実環境
観測期間	–	Feb 2017 – Feb 2018
データ数	1,208	184,599
送信元 IP アドレス数	–	10,138
送信先 IP アドレス数	–	848

は D_{lab} と D_{real} を用い，性能評価には D_{real} を用いた．

D_{lab} では検知漏れ評価および誤検知評価のために，2 種類の評価用データ (攻撃成功時データ，攻撃失敗時データ) を以下の手順で用意した．ここで言う検知漏れとは成功した攻撃を失敗と判定したことを意味し，誤検知とは失敗した攻撃を成功と判定したことを意味している．判定不可の場合は検知漏れ，誤検知としては扱わない．

まず，公開されている攻撃コード [81, 82] および実環境で観測した攻撃コードを収集した．次に，攻撃コードの意味としては同様であるが，乱数等が攻撃コードに用いられている場合を集約した．表 5.4に集約後の例を示す．集約後の攻撃コードは 7,819 個であった．集約する理由は，類似する同様な攻撃コード数の偏りによる精度の変動を避けるためである．これらの攻撃コードが成功した際に出力されると思われる痕跡を手動で付与できた攻撃コード 1,208 個を検証用攻撃データとした．また検証用攻撃データを実行した際の出力を攻撃成功時データとし

表5.4: 集約後の攻撃コードの例

集約前	集約後

`echo GBJIVY$((45+22))`
`echo GEWHYE$((65+76))`
`echo HWUZPQ$((82+33))`
	`echo GBJIVY$((45+22))`

`sleep(5)` `sleep(10)`	`sleep(5)`
`../../etc/passwd` `../../../etc/passwd`	`../../etc/passwd`

た．また，攻撃失敗時データは，Alexa Top 50 [83] ドメインの Web ページを結合したものを利用した．出力を付与できない攻撃コードは`sleep`のような時間が差異になるものや，`wget`のような外部に通信を行う攻撃である．本研究ではレスポンスに痕跡が現れる攻撃のみを対象としているが，処理時間や通信の試行も対象とすることでより対応できる攻撃が増える可能性があり，今後の課題である．

検知漏れ評価では，検証用攻撃データをエミュレートし，攻撃成功時データと合致する IOC を提案システムが出力できたか否かで評価を行う．IOC を出力できなかった場合，検知漏れとして扱う．誤検知評価では，検証用攻撃データをエミュレートし，出力された IOC が攻撃失敗時データに合致する IOC が出現するか否かで評価を行う．IOC が現れた場合，誤検知として扱う．D_{lab} に含まれる攻撃カテゴリ毎の検証用攻撃データ数と攻撃コードの概要を表 5.5に示す．D_{real} の攻撃検知にはチューニングされた WAF の検知結果および，独自シグネチャを用いて判定を行った．

評価の際は最小 IOC 文字数の閾値 $T = 5$ として実験を行った．この理由は，$T = 5$ の場合が提案手法の効果を最大限に発揮する可能性が高いためである．

5.5.1　精度評価

D_{lab} を用いて精度評価を行った．評価には攻撃成功に対する適合率 (Precision)・再現率 (Recall) および，攻撃失敗に対する適合率・再現率の 4 つの指標を用いた．ここでいう適合率とは判定可能と判断したデータに対する判定結果が正しいデータの割合である．また，再現率は全データの指標ラベルに対して，同じ判定結果を出した場合の割合である．つまり，以下の

表5.5: 検証用攻撃データの攻撃カテゴリ別データ数

カテゴリ	個数	説明
RCE	217	`cat`, `ls`, `uname`, `whoami`, `last`, `netstat` 等の情報を出力するコマンドの実行
SQLI	222	SELECT 文による DB のバージョン，ユーザ名，DB 名，ホスト名，テーブル名の取得
PT	136	Linux や Windows の設定ファイル等 (`passwd`, `win.ini`) の取得
XSS	633	`script`, `img`, `body`, `a`, `embed`等のタグの挿入，JavaScript コードの挿入

表5.6: 判定結果

指標 / 判定結果	攻撃成功時データ	攻撃失敗時データ
攻撃成功	868	108
攻撃失敗	0	760
判定不可	340	340
計	1208	1208

通り定義できる．

$$\text{適合率} = \frac{\text{判定結果が正しいデータの個数}}{\text{判定可能なデータの個数}} \tag{5.1}$$

$$\text{再現率} = \frac{\text{判定結果が正しいデータの個数}}{\text{全データの個数}} \tag{5.2}$$

D_{lab} に対する判定結果を表 5.6に示す．また，判定結果から算出した適合率および再現率の結果を表 5.7に示す．成功判定に対する適合率は 100% であるから，提案手法で攻撃が成功したと判定できた場合，その結果は正しく，重要なアラートであることを意味する．またデータセット全体に対しても 93.8% の適合率を示すことから，判定可能な攻撃については高い精度で成否が判定できている．一方，再現率は成功判定・失敗判定どちらも約 60% 以上であった．つまり，セキュリティオペレーションを行う分析者にとっては対応すべきアラートが 60% 以上削減されるため，稼働削減につながる．

次に，D_{real} に対する評価結果を表 5.8に示す．成功と判定した件数は 190 件であった．190 件のアラートについてその正しさを目視で確認した結果，攻撃成功と確認できたのは 90 件で

表5.7: D_{lab} に対する適合率および再現率

項目		割合	
適合率 (Precision)	攻撃成功	100%	868/868
	攻撃失敗	87.5%	760/868
	全体	93.8%	1628/1736
再現率 (Recall)	攻撃成功	71.9%	868/1208
	攻撃失敗	62.9%	760/1208
	全体	67.4%	1628/2416

表5.8: D_{real} に対する評価結果

判定結果	個数	割合 (%)
成功	190	0.10
失敗	96,223	52.1
判定不可	88,186	47.8

あり, 残り 100 件は誤検知であった. この結果は, 提案手法が調査すべきアラート数を 184,599 件から 190 件までに絞り込むことができたことを意味する. 調査すべきアラート数を大きく減少させれることから, WAF や IDS のアラートでは気付きにくい成功した攻撃を発見できることを示している. 攻撃成功と確認できた 90 件の攻撃は重複しており, 実際は 3 種類の攻撃であった. 誤検知の原因は 5.5.3節にて述べ, 検知した事例は 5.5.4節にて紹介する. 判定不可となった攻撃 88,186 件を詳細に分析し, 原因を 5 つに大別できた. 表 5.9にその結果を示す. 最も大きな原因は出力がない攻撃である. これらの攻撃は脆弱性を確認するための処理時間の差を利用した攻撃であったり, サーバ内の設定を変更する攻撃などであった. 例えば, 以下の攻撃コードの場合,

```
echo 'Hacked' > /var/www/index.php
```

攻撃が成功しても出力はないが Web サイトが改ざんされることとなる. 攻撃した際に出力がないため, 痕跡が抽出できず判定不可となった. 次に原因として多い, 攻撃でないリクエストは誤って WAF 等に検知された可能性が高いものであり, 攻撃コードが抽出できず判定不可となった. 未対応のエミュレーション環境への攻撃は Windows の設定ファイルや特定の Web アプリケーションの設定ファイルを漏洩させる攻撃であった. 今回の実験ではエミュレータに利用する OS は Linux が前提であり, また特定の Web アプリケーションに特化していないため, 判定不可となった. 外部通信を行う攻撃は`wget`や`curl`を利用して攻撃に用いるスクリプトファイル等のダウンロードを行う攻撃であった. 提案手法の実装では外部に通信を許可して

表5.9: 判定不可となる原因

原因	割合 (%)
出力が存在しない攻撃	55.7
攻撃ではないリクエスト	27.2
未対応のエミュレーション環境への攻撃	14.7
外部通信を行う攻撃	2.37
その他	0.03

いないため，スクリプトファイルの内容確認できず，判定不可となった．

5.5.2 性能評価

性能評価では各攻撃リクエストの検証に要する時間を測定した．性能評価に利用したマシンのスペックは 1.3 GHz Intel Core i5 4250U CPU, 8GB RAM, 512GB SSD である．図 5.3に測定結果を示す．約 98% の攻撃リクエストに対する処理が 1 秒以内に終了している．つまり，WAF と Web サーバとの間にインラインで配置して，攻撃が成功したと判定できた場合に遮断を行うという運用をした場合でも，1 つのリクエストに対するレスポンスを約 1 秒程度の遅延に抑えられることを意味する．WAF にて攻撃と判定されるリクエストは、一般に全リクエストのうちのごく一部に限られる．したがって，そのような限られたリクエストを処理する場合のみ 1 秒程度の遅延が発生するのであればユーザビリティは大きく損なわれない [64] ため，性能面でも実用的であると考える．

5.5.3 分析

誤判定の原因

正しく判定できなかった原因の分析を行った．D_{lab} による評価の結果，提案手法は成功した攻撃を失敗と判定することはなかった．また，D_{real} の攻撃失敗と判定された通信のレスポンスに対して D_{lab} で攻撃成功と判定した際の IOC を照合させたところ，合致するものは存在しなかった．公開されている攻撃コードは頻出するものであるから，D_{real} 中の攻撃が攻撃失敗と判定されている中に含まれている可能性は低いと考える．つまり，攻撃失敗と判定した場合はそのアラートは分析者で再度確認する必要がない信頼性を有することを意味しており，提案手法はオペレーションの効率化に寄与できる．成功した攻撃を失敗判定してしまう場合，結局分析者は再度，失敗判定された全ての攻撃のアラートを確認し，成功した攻撃を発見せねばならず，セキュリティオペレーションの効率化にならない．

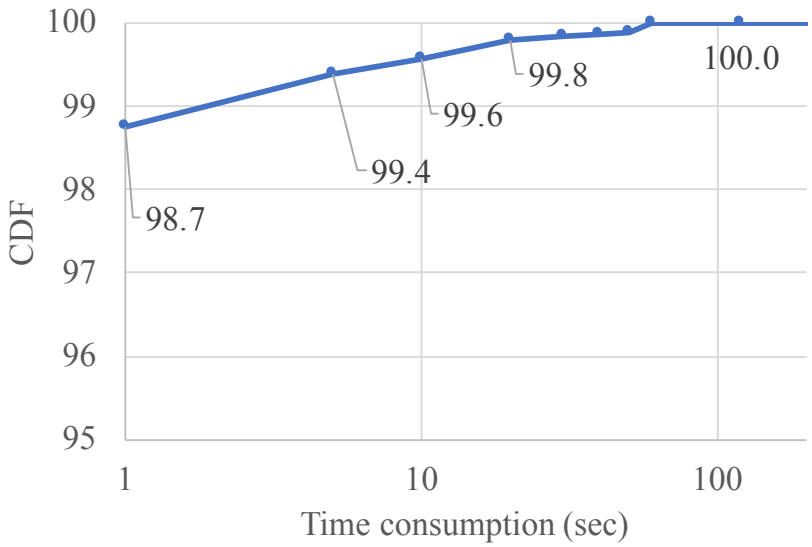

図5.3: 処理時間の累積分布関数

D_{lab} で評価した際，失敗した攻撃を成功と判定した件数は 108 件であった．その原因は IOC の正規化処理が不十分で，IOC に汎用的な表現が残ってしまい，一般的な Web ページの内容にも現れる表現を含んでしまったためである．例えば，SQL インジェクションで`select version()`といった攻撃コードの場合，MySQL ではバージョンの文字列を表す`\d+.\d+.\d+`といった IOC が抽出される．これは一般的な Web ページにもよく現れる表現であったため，攻撃が成功したと誤って判定してしまっていた．

また 5.5.1節で述べた D_{real} で評価した際の 100 件の誤検知原因は，上記の IOC が汎用的である以外に，レスポンスの内容をリクエストの際のボディ部に含めてリクエストを行うアクセスの存在にもある．例えば，サーバからのレスポンス内容が`<html>Hello World</html>` のような固定値であり，この内容をリクエストのボディ部に設定してアクセスした場合，提案手法ではリクエストの内容がそのまま表示されていると判定し，XSS による攻撃が成功したと判定する．しかし，実際にはレスポンスは固定値のため，攻撃者によって任意のコードを挿入されてないが，攻撃が成功したと誤検知してしまった．

最小 IOC 文字列長 T による精度の変化

提案手法では IOC を抽出する際，最小 IOC の文字列長として閾値 T を設けている．T の変動による適合率，再現率の変化を図 5.4，図 5.5にそれぞれ示す．適合率に関しては攻撃成功および攻撃失敗ともに頂点が存在する．特に $T = 5$ の時は，攻撃成功の適合率が 100% になり，成功した攻撃を見逃すことがなくなる．再現率に関しては，攻撃成功する場合は頂点 $(T = 10)$ が存在する．以上のことから，T は $[5, 10]$ の間に設定するのが適切である．

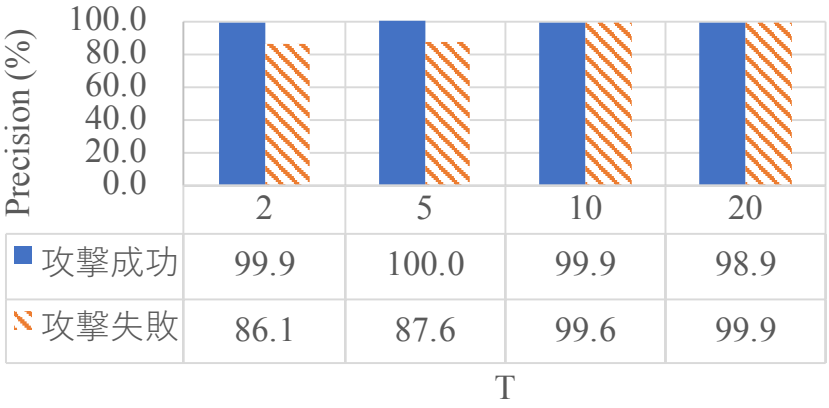

図5.4: T による適合率の変化

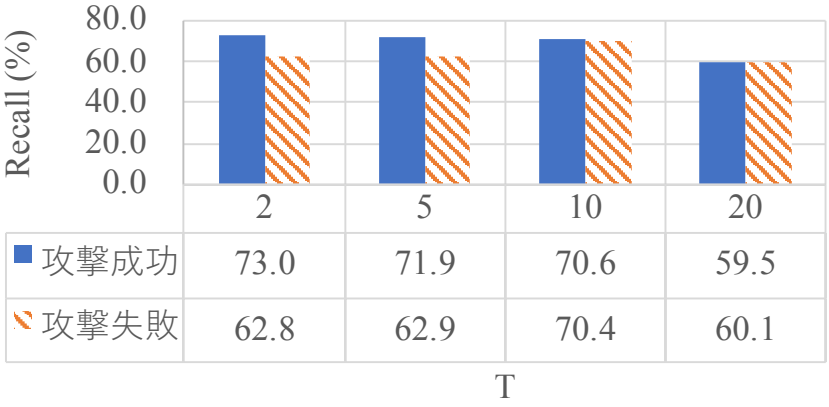

図5.5: T による再現率の変化

表5.10: T による IOC 数の変化

T	2	5	10	20
全 IOC 数	78,662	76,990	68,766	50,134

T の増加に対して，適合率・再現率が単調増加あるいは単調減少にならない理由は IOC の数によるものである．表 5.10より，T と全 IOC 数が反比例していることがわかる．全 IOC 数とは全攻撃コードから抽出した IOC の数である．T が $[0, 5)$ と小さ過ぎる時は不適切な IOC が除外されないため，精度が低い．逆に T が $(10, 20]$ と大き過ぎる時は正しく判定するのに寄与する IOC が除外されてしまい，精度が落ちてしまった．

```
UNION SELECT CHAR(45,120,49,45,81,45),CHAR(45,120,50,45,81,45),CHAR(45,120,51,45,81,45),CHAR
(45,120,52,45,81,45), ... -- /*
```

図5.6: SQL インジェクションの攻撃コード

```
  <table>
    <tr><th class="style_th">entry1</th><td class="style_td"> -x5-Q- </td></tr>
    <tr><th class="style_th">entry2</th><td class="style_td"> -x6-Q- </td></tr>
    ...
    <tr></tr>
  </table>
```

図5.7: SQL インジェクション攻撃に対する HTTP レスポンス

5.5.4　事例紹介

D_{real} に対して攻撃成功と判定できた事例について紹介する．今回の実験で攻撃成功と判定できたリクエストは 90 件であるが，攻撃カテゴリ別に分類すると以下の 3 つの事象に分けることができた．事例を示すにあたって，実システムの特定を避けるため，表現を一部置き換えている．

- **事例 1：** SQL インジェクションの脆弱性有無を確認する攻撃
- **事例 2：** WordPress の設定ファイルの漏えい
- **事例 3：**ログインフォームに対する XSS

事例 1 では図 5.6に示す攻撃コードが送られていた．この攻撃コードが実行された場合，攻撃コードの SELECT 文が指定した値が出力される．攻撃コードでは CHAR 関数によって難読化が施されているが，例えば`CHAR(45,120,49,45,81,45)`は文字列`-x1-Q-`を表す．つまり，SQL 文が実行された場合は上記の CHAR 関数が実行されて，その結果の文字列が出力される．図 5.7に示すこの攻撃に対する HTTP レスポンスを見ると，SQL 文を実行した後の文字列が出力されているので，攻撃が成功していると判定した．今回の攻撃コードは脆弱性の存在を調査するものであり，被害を及ぼすことはないが，パスワードを漏えいさせる等の攻撃に容易に転用できるため重大インシデントにつながる．

事例 2 では，攻撃者は図 5.8に示すように WordPress 設定ファイルのバックアップが置かれていないかをチェックする攻撃を行っている．これに対して HTTP レスポンスでは図 5.9に示す内容を返しており，設定ファイルが漏えいしていると判定した．設定ファイルにパスワードなどの機微情報が設定されている場合，パスワード漏えいという重大インシデントにつなが

```
1 GET //wp-config.php~ HTTP/1.1
```

図5.8: WordPress 設定ファイルのバックアップに対するリクエスト

```
1 /* WordPress Database Table prefix. */
2 $table_prefix  = 'wp_';
3 define('WP_DEBUG', false);
4 if ( !defined('ABSPATH') )
5   define('ABSPATH', dirname(__FILE__) . '/');
```

図5.9: WordPress 設定のバックアップファイルアクセスのレスポンス

```
1 GET /*.asp?lang="><script >alert(String.fromCharCode(88,83,83))</script> HTTP/1.1
```

図5.10: ログインフォームに対する XSS 攻撃

```
1 <form method="POST" action="*.asp" name="login">
2 <input type="hidden" name="lang" value="
  "><script >alert(String.fromCharCode(88,83,83))</script> ">
3 </form>
```

図5.11: ログインフォームに対する XSS 攻撃のレスポンス

ってしまう．

事例 3 では図 5.10，図 5.11に示すように攻撃者はある ASP スクリプトの`lang`というパラメタに`script`タグを挿入している．挿入されたタグは HTTP レスポンスにそのまま出力されているため，攻撃が成功したと判定した．

これらの事例は提案手法を導入して初めて発見したものであり，WAF アラート通知では調査・分析まで至らなかった．

5.6 制約

アプローチの特性上，提案手法には以下の攻撃の成否を判定できないという制約が存在する．

出力が存在しない攻撃 提案手法では攻撃が成功したと判定できる指標を用いることで攻撃成否を判定する．しかし，攻撃によっては成功しても指標が出力されない，あるいは存在しない場合もある．

未対応のエミュレーション環境への攻撃　提案手法では可能な限り対象サーバと同等のシステム環境を用いて攻撃をエミュレートする．しかし，特別な製品を用いている場合，エミュレーション時に IOC を抽出できても，対象のシステム環境の出力が全く異なるため，正しく判定できない．

外部通信を行う攻撃　提案手法では外部への攻撃を避けるためにエミュレーション時はネットワーク通信を遮断している．そのため，攻撃コードが外部のリソースを取得してから実行される場合，正しく判定できない．

5.7　関連研究

著者らが既存の研究を調べた限りでは既存の攻撃に対するその成否を判定する手法は，大きく 4 つのアプローチに分けられる [84].

HIDS は OS やアプリケーションが出力する情報に基づいて攻撃検知を行うため，攻撃を正しく検知した際は攻撃が成功した後という可能性が高い．そのため，HIDS がアラートを通知している際は攻撃を成功したと判定していると考えることができる．システムコールの順序や種類を特徴とする検知手法 [33] が一般的であるが，システムコールのみならず，DB アクセスなどのアプリケーションレイヤーの情報も用いる検知手法も提案されている [70]．導入にはシステムコールなどを観測する仕組みが必要であるため，商用環境では動作保証が問題となり導入障壁が大きい．

攻撃があった際，予め定義された状態に至ったかどうかで攻撃の成否を判定するアプローチも存在する [85]．例えば，Vigna らの手法 [74] では攻撃の状態遷移に基づいて攻撃の成否を判定する．Robin らの手法 [75] および Zhou らの手法 [76] は既存の IDS のシグネチャを活用して，シグネチャマッチをネットワーク通信に対して複数箇所で行うことで，攻撃の成否を判定する．また脆弱性自体の処理を捉える研究 [86, 87] も存在する．問題点として，攻撃状態をルールで定義したり，アプリケーションの脆弱な箇所の動作をルールで定義したりする必要があり，利用者に要求される知識レベルが高い点が挙げられる．ルールを自動的に生成する手法 [88, 89] も提案されているが，様々なアプリケーションに対する攻撃の成否ラベルを付与した通信の学習データを用意することも必要である．提案手法では利用者に攻撃カテゴリの固有表現を求めるが，入手が容易な OS コマンドの一覧や関数の一覧等を設定するのみであり，このアプローチが求める攻撃の状態や脆弱性の処理方法といった攻撃や脆弱性に関する特有な情報ではない一般的な情報であるため，比較的低い知識レベルの利用者でも運用可能である．

IDS のアラートと脆弱性スキャナの情報を関連づけることで攻撃対象のサービス・アプリケーションが脆弱性を持っているかどうかで攻撃の成否を判定するアプローチも存在する [90, 91, 92]．例えば，Kruegel らの手法 [90] では IDS のアラートに含まれる CVE 番号

と脆弱性スキャナで発見した脆弱性の CVE 番号が同一かどうかで関連付けを行う．この手法では Web アプリケーションや Web サーバに脆弱性が存在するバージョンを利用していないか確認し，その脆弱性に対する攻撃を検知した場合は有効な攻撃と判定する．近年の SIEM (Security Information and Event Management) 製品 [93, 94, 95] でも同様の判定アルゴリズムを実装しているものも存在する．このアプローチと同様に脆弱性スキャナのルールの更新，IDS シグネチャとの関連付けのための情報の付与が必要になり，多様な攻撃に対応することが難しい．

攻撃があった際に，その攻撃コードの挙動をエミュレートし，実行する機械語命令列を抽出するアプロートも存在する [71, 72, 96]．このアプローチではホスト内でも同じ機械語命令列の実行を観測した場合，攻撃が実行されたとしてアラートを出力することで攻撃が成功したときのみを判定することが可能となる．これらの手法は x86 アセンブリの攻撃コードのエミュレーションに限られるが，Web アプリケーションに対しては，コードインジェクションの攻撃に特化した WeXpose [73] が存在する．しかし，これらの手法ではエミュレーション後の挙動を実システム内で観測する必要があるため，HIDS と同様に導入先の改変が必要である．

本研究はシステム改変が不要および比較的低い知識レベルの利用者でも運用可能という 2 つの観点で既存研究と異なる．

5.8 結語

大量のアラートから重要なインシデントに関わるアラートを人手で探し出すには多くの時間を要する．本研究では，攻撃の成否に応じてアラートの重要度を決定する手法を提案した．提案手法では攻撃コードのエミュレーションを行い，攻撃の痕跡である IOC を抽出する．IOC が HTTP レスポンスに含まれるか否かで攻撃の成否を判定し，アラートの重要度を決定する．評価より 67.4% の割合 (再現率) の Web 攻撃に対して，93.8% の精度 (適合率) で判定可能であることを示した．また，約 98% の攻撃のエミュレーションにかかる時間が 1 秒以内であることから，実用的な結果が得られた．

第 6 章

リモートシェルコードのエミュレーションに基づく攻撃の成否判定手法

6.1 緒言

5章で提案した手法は OS コマンドや SQL といった人間に理解しやすい平文による攻撃を想定し，攻撃の痕跡がレスポンスに現れる前提とした判定手法である．しかし，平文による攻撃でなく，シェルコードと呼ばれるプログラムのバッファやヒープなどのメモリ領域をオーバーフローさせる脆弱性を狙った機械語命令列による攻撃も存在し，攻撃の痕跡がリクエストに対するレスポンスではなく，他の通信に現れる場合も存在する．本研究ではこのようなリモート型シェルコードによる攻撃に対して攻撃の成否を判定する手法を提案する．5章との差異を表 6.1に示す．

本研究では IDS のアラートを引き起こす攻撃の一つの要因であるシェルコード [97, 72] に着目する．ここでいうシェルコードとは攻撃者が攻撃対象サーバを制御するために使用する機械語の命令列である．攻撃者が挿入するシェルコードのエミュレーションを行い，エミュレーション時に観測する動作から攻撃の痕跡を抽出する．攻撃の痕跡に含まれる通信の挙動と実際の通信を比較することで攻撃の成否を判定する．分析者に攻撃の成否に関する参考情報を与えることでアラート分析の効率を向上させる．

表6.1: 5 章の提案手法との差異

攻撃種類	痕跡の箇所	
	レスポンス	外部通信
平文攻撃	5 章	
シェルコード		6 章

```
1  push    0x6603a8c0      ; ipaddr = 192.168.3.102
2  push    0xefb0          ; port = 45295
3  push    2
4  mov     ecx, esp
5  push    0x10
6  push    ecx
7  push    eax
8  mov     ecx, esp
9  mov     esi, eax
10 push    3               ; set connect
11 pop     ebx
12 push    0x66            ; set socketcall
13 pop     eax
14 int     0x80            ; syscall
```

図6.1: シェルコードの逆アセンブル結果の例

本研究の貢献は以下の通りである．

- リクエストに対するレスポンスに攻撃の痕跡が現れる攻撃に対応した既存研究 [69] を拡張し，レスポンス以外の通信に攻撃の痕跡が現れる攻撃にも対応する成否判定手法およびその実装を提案した．
- 評価により，通信に特徴が現れる攻撃に対して 60% 以上を正しく判定できた．
- 実際に行われたセキュリティコンテストの通信に対して手法を適用し，攻撃の成否を自動的に判定できた事例が複数存在した．

6.2 背景

6.2.1 シェルコード

本稿では、シェルコードを攻撃者が攻撃対象サーバを制御するために使用する機械語の命令列と定義する．つまり，可読文字列からなるスクリプトコードや OS コマンドにより侵害ホストの制御を奪うコードは本研究の対象外とする．図 6.1に本研究で対象とするシェルコードを逆アセンブルした例を挙げる．このシェルコードでは 1,2 行目で接続先の IP アドレスおよびポート番号を指定し，14 行目のシステムコール発行命令により，通信を行う．シェルコードはローカル型，リモート型の 2 種類に大別される [98]．ローカル型は攻撃者が攻撃対象のサーバのシェル・ターミナルをすでに制御しており，権限昇格等を狙う際に使用される．リモート型は攻撃者がネットワークを介してサーバを攻撃する際，TCP/UDP 通信を攻撃者が用意したサーバに行わせることにより攻撃対象サーバを制御する際に使用される．リモート型のシェルコードはその通信方法により更に connect-back, bindshell, socket-reuse の 3 種類に分かれる．connect-back の場合は攻撃対象サーバから攻撃者のサーバへ新たに接続を行う場

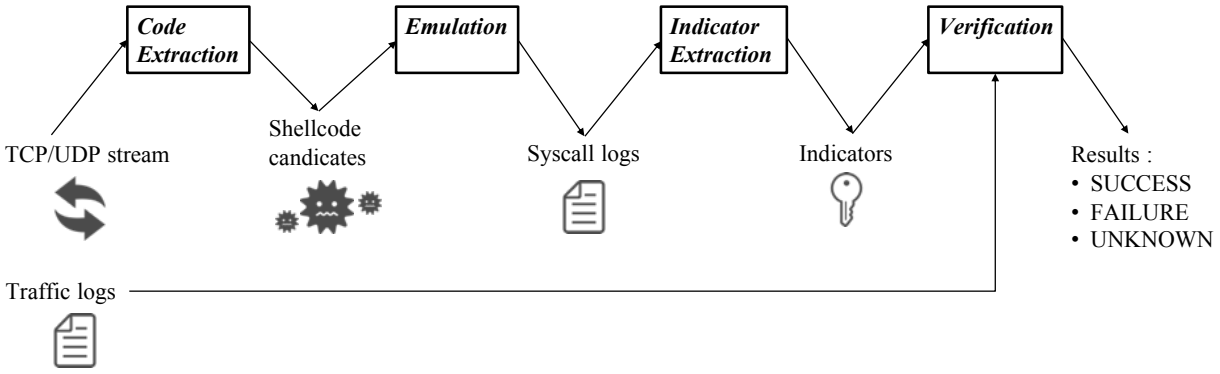

図6.2: 提案手法の概要

合，bindshell は攻撃者のサーバから攻撃対象サーバへ新たに接続を行う場合を意味している．socket-reuse は攻撃する際の通信をそのまま制御に利用するので新たな接続を行わない場合を意味している．socket-reuse の場合は攻撃時に現在の通信が切断されないようにする必要があるため実際に利用されることは限定的である．

6.2.2 攻撃の成否

サーバに対する攻撃の結果は，成功と失敗に大別される．攻撃の成功とは，攻撃者の意図通りに攻撃が行われている状態のことである．図 6.1の場合，攻撃が成功すると IP アドレス 192.168.3.102 のポート番号 45295 に対して接続を行う．そのため，ネットワーク通信のアクセスログに攻撃コードが示す接続先との通信が記録されていた場合，攻撃は成功していると判断できる．一方，攻撃が失敗している場合には挿入されたシェルコードは実行されないため，攻撃コードが示す接続先への通信は発生しない．したがって，アクセスログに攻撃コードに含まれる接続先とのログが含まれていなければ攻撃は失敗したと判断できる．本研究では攻撃成否判定の結果を IDS 等のアラートに付与し，一様だったアラートに対して優先度付けを行う．これにより，侵害等が起きた際に迅速に対応可能にすることを目的としている．

6.3 提案手法および実装

本研究ではリモート型のシェルコードによる攻撃の成否を判別する手法を提案する. 図 6.2に提案手法の各処理の概要を示す.

提案手法には 4 つのステップがある．最初のステップである **Code Extraction** では TCP/UDP ストリームからシェルコードである可能性が高いバイト列を抽出する．次に，抽出したシェルコードをエミュレータを用いて実行する (**Emulation**) ．エミュレーションの際，シェルコードの挙動としてシステムコールを観測する．次に観測したシステムコールから

攻撃の痕跡として利用できるものを選出する (**Indicator Extraction**) ．最後に，選出された痕跡を活用し，実際の環境で観測したトラフィックと比較することで攻撃の成否を判定する (**Verification**) ．

6.3.1 Code Extraction

攻撃と検知された TCP/UDP ストリームを入力として，そのストリームからシェルコードの抽出を行う．抽出方法は以下の 3 つである．

1. 既知シグネチャの検知による抽出
2. NOP スレッドの有無を用いた抽出
3. GetPC コードの有無を用いた抽出

1 番目の抽出方法では IDS の検知シグネチャを公開しているサイトである Emerging Threats [46] 等から収集した．2,3 番目の抽出方法は既存研究 [99, 100, 101] で報告されているシェルコード部分の特徴を利用した．

NOP スレッドとは何も処理をしない命令列のことである．バッファオーバーフローなどの攻撃の場合，オーバーフローした際に実行される命令が参照できないと，セグメンテーション違反が起き，攻撃者が用意したシェルコードは実行されない．そのため，攻撃者はセグメンテーション違反等を回避するためにメモリアドレス空間を NOP 命令で埋めることにより，用意したシェルコードを実行させる可能性を高める．NOP スレッドは x86 CPU アーキテクチャの場合，バイト値`0x90`等が連続するといった特徴的があるため，本手法はその特徴を基に NOP スレッドの検出を行う．NOP スレッドが検出された場合，NOP スレッドの終点を起点として，その後に続くバイト列をシェルコードとして抽出する．抽出するシェルコード終点は TCP/UDP ストリームの終点である．

シェルコードによっては自身の一部を難読化しておいて，実行時のみに復号して実行するものも存在する．復号には暗号化した部分のメモリアドレスの絶対アドレスが必要になるが，このアドレスを算出するために GetPC コードと呼ばれる命令列を実行することが多い．本手法はその特徴に着目し，GetPC コードよく利用される FSTENV 等の命令列のバイト列を検出し，そのバイト列を含む機械語命令として有効な可能な一連のバイト列を抽出する．抽出するシェルコードの起点は GetPC コードの開始位置より以前を逆順に走査して GetPC コードまで機械語命令・メモリ違反なく命令列を実行できる最長の位置がが起点となる．機械語命令・メモリ違反の判定には CPU エミュレータである Unicorn [102] を用いた．抽出するシェルコード終点は NOP の場合と同様に TCP/UDP ストリームの終点である．

シェルコードを抽出できない場合は判定不可の結果を出力し，処理を終える．

6.3.2 Emulation

前処理で抽出したシェルコードのエミュレーションを行う．アーキテクチャの差異によってエミュレーションできず，攻撃による痕跡を観測できない場合を減らすため，x86，x86-64，ARM など様々な環境を用意する．エミュレーションではシェルコードのバイト列をメモリに読み込み，読み込んだバイト列のメモリアドレスから実行を開始するプログラムを用意して，そのプログラムを実行する．シェルコードによって発生する挙動を観測するため，シェルコードを実行しているプロセスが発行するシステムコールを監視する．プロセスが終了した場合はシステムコール監視を停止する．接続待ちや攻撃者による意図的な遅延によってエミュレーションが終了しない事態を避けるため，エミュレーションにはタイムアウトを設ける．

6.3.3 Indicator Extraction

前処理によって観測したシステムコールの情報を基に攻撃の成否判定に利用できる痕跡を抽出する．提案手法ではネットワーク通信から観測可能な以下の 4 つのシステムコールの発行を攻撃の痕跡として記録する．

- connect, sendto システムコール
  外部に対して接続を行う挙動を表す．接続先 IP アドレスおよびポート番号を痕跡として記録する．
- bind, recvfrom システムコール
  外部からの接続を受け付ける挙動を表す．攻撃対象サーバの IP アドレスおよびポート番号を痕跡として記録する．

TCP 通信であれば, socket システムコール後に connect/bind システムコールを発行する必要があり,UDP 通信も,一般的には connect/bind システムコールを利用するが,sendto/recvfrom システムコールでもデータの送受信が可能なため，この 4 つのシステムコールを監視対象とした．リクエストに対するレスポンスに攻撃の痕跡が現れる場合はすでに既存研究 [69] で取り扱われているため，本稿では対象外とする．IP アドレスおよびポート番号に加えて通信の内容も痕跡として，同じ内容を送受信しているかを判定することでより確実な成否判定が可能となる．例えば，TCP セッションの確立後，シェルコードが ABCD の文字列を送信することがわかっていれば，この ABCD という文字列の存在の有無によってより確度高く成否判定できる．しかし，攻撃者は容易に通信の内容を暗号化することが可能なため，検出を回避される恐れがある．そのため，提案手法では動作，IP アドレスおよびポート番号のみを攻撃痕跡の情報として利用した．抽出する痕跡の例を表 6.2に示す．

表6.2: 抽出した攻撃痕跡の例

動作	IP アドレス	ポート番号
connect, sendto	192.168.1.2	4444
bind, recvfrom	192.168.1.1	4444

痕跡が抽出できない場合，攻撃の成否を判断することができないことから，判定不可の結果を出力し，一連の処理を終える．

6.3.4 Verification

前処理によって得られた痕跡を基に実際の通信が攻撃の痕跡と同じであるかか否かを判定する．痕跡の動作が connect, sendto の場合，抽出した IP アドレスおよびポート番号に接続を行う通信，つまり TCP の SYN パケットあるいは UDP のパケットが送信されたか否かによって判定を行う．送信された場合，攻撃は成功，送信されていない場合に攻撃は失敗と判定する．

動作が bind, recvfrom の場合，攻撃が行われたサーバの IP アドレスおよび痕跡として抽出したポート番号への接続が確立したか否かによって判定を行う．TCP を利用した通信の場合，攻撃者のクライアントから SYN パケットが抽出した IP アドレスおよびポート番号に送信された場合，SYN-ACK を返送したか否かによって判定する．UDP を利用した通信の場合，ICMP Port unreachable が攻撃者に返送されたか否かによって判定を行う．ICMP Port unreachable を返送した場合，ポートは利用できる状況でないため，攻撃は失敗したと判定する．返送しない場合攻撃は成功したと判定する．

通信の観測には一定時間 T が設けられており，時間 T 以内の通信記録から判定する．T はユーザが指定する閾値である．閾値 T を設けた理由は，通信先が通常のアクセスでも行う IP アドレスとポート番号の場合，誤判定になってしまうため，時間制約により判定の正確性を維持するためである．

時間 T 以内に接続を行う通信を観測できなければ，シェルコードは実行されていないため，攻撃は失敗したと判定する．

6.3.5 実装

提案手法を実装したシステムの全体像を図 6.3に示す．TCP/UDP ストリームの取得には OSS のネットワーク監視ツールである Zeek (Bro) [80] を利用した．攻撃成否の判定の部分は Python で実装した．攻撃コードをエミュレートする環境には Docker [79] および既存の CPU エミュレータである libemu [103] を採用した．libemu は x86 のシェルコードのみに対応して

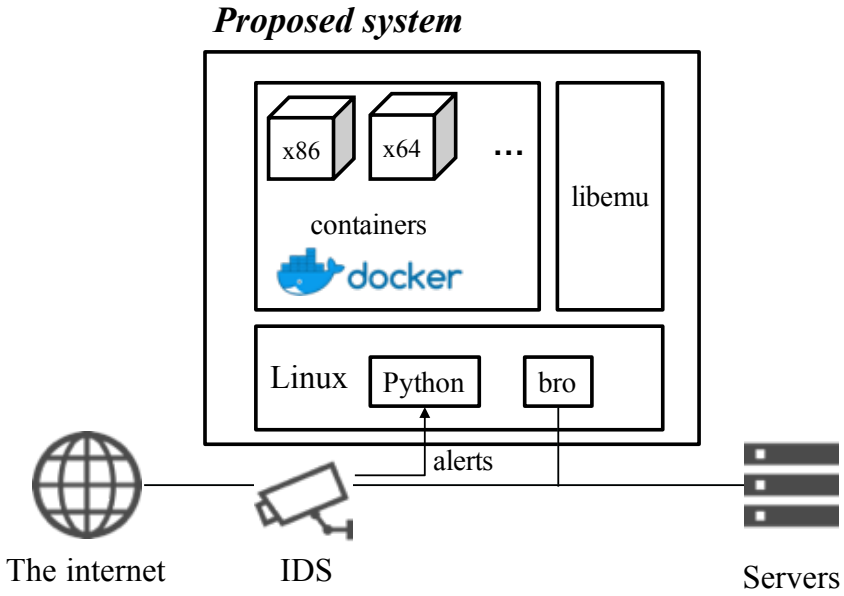

図6.3: 提案システムの概要

いるため，x86 のシェルコードの場合のみ追加で libemu でもエミュレーションを行う．CPU エミュレータはより詳細にシェルコードを分析できる利点があるが，シェルコードを動作させるためには正しくメモリ空間をする必要がある．また，システムコールや API をエミュレートする訳ではないため，利用者で適切に定義する必要があるという難点がある．Docker を用いたエミュレータの場合，Docker のみではシェルコードを詳細に分析できないが実態により則した環境を用意できることが利点である．そのため，提案システムでは Docker と libemu を組み合わせて利用する．提案システムでは攻撃が発生する度，CPU エミュレータである libemu によるエミュレーションおよび，エミュレータとなる Docker Container を作成し，その環境内でシェルコードを実行する．なお，Docker によるエミュレーション時は外部への攻撃を防ぐため，ネットワーク通信は全て遮断した．

6.4 評価

精度および性能の 2 つの観点から評価を行った．

6.4.1 精度評価

精度評価では複数のシェルコードを用意し，攻撃が成功した際，提案手法が正しく攻撃成功判定ができている割合 (再現率) ，提案手法によって成否判定が可能な際に正しく攻撃成功が判定できている割合 (適合率) および，仮に IDS が誤検知した際に正常の通信をシェルコード

表6.3: D_{msf} 生成に用いた MSFvenom のオプション

オプション	値
エンコード方法	無，`add_sub`，`alpha_mixed`，`bloxor`，`shikata_ga_nai`，等 (22 種類)
エンコード回数	1, 2
回避文字列	無, `\x00\xff`
NOP スレッド	無, 100

による攻撃として判定してしまう割合 (誤判定率) を調べた．評価ではバッファオーバーフローの脆弱性があるサーバプログラムを用意し，シェルコードを挿入することで攻撃を再現した．なお，攻撃を再現する際に NOP スレッドを用いているため，シェルコードが正しく抽出できている状態である．

再現率および適合率を評価するデータセットは 2 種類用意した．1 種類目のデータセット D_{edb} は様々なシェルコードに対応できるかという観点で，シェルコードのデータベースである Exploit-DB [104] から収集したシェルコード 898 件を用いて作成した．今回の評価では Linux OS を対象とした x86 および x86-64 アーキテクチャに対するシェルコード及び攻撃の痕跡が通信に残るシェルコード 115 件のみを対象とした．2 種類目のデータセット D_{msf} はもとは通信の接続を行う 1 つの x86 用シェルコードであるが，様々な変換・難読化を施すことにより生成した複数のシェルコードを用いた．変換・難読化には Metasploit [105] の機能である MSFvenom を利用し，変換・難読化に成功したもののみを用いた．表 6.3にその際に利用したオプションを示す．

誤判定率の評価に用いるデータセットには，Wireshark のサイト [106] から収集した様々なプロトコルの通信が含まれる 779 個の PCAP ファイルから作成した．収集した PCAP ファイルから 87,774 件の TCP/UDP ストリームを抽出し，各ストリームを提案手法で攻撃として判定してしまうか否かを調べた．表 6.4に評価結果を示す．D_{edb}, D_{msf} どちらともに 60% 以上のシェルコードから痕跡を抽出でき，正しく攻撃が成功したと判定できていた．シェルコードから痕跡を抽出できなかった理由はシェルコードの実行に失敗してたためである．この原因は大きく 2 つであった．1 つ目はシェルコードが別の OS コマンド，例えば wget や nc を呼び出して外部通信を行う仕組みになっていたことである．エミュレータではそれらコマンドインストールしておらず，差異が発生し攻撃成功を正しく判定できなかった．2 つ目はエミュレーション時，シェルコードが正しい番号のシステムコールの番号を計算できなかったことにある．

表6.4: 精度評価結果

項目	データセット	割合
再現率	D_{edb}	60.0% (69/115)
	D_{msf}	64.8% (70/108)
適合率	D_{edb}	100% (69/69)
	D_{msf}	100% (70/70)
誤判定率		0.01% (5/87,774)

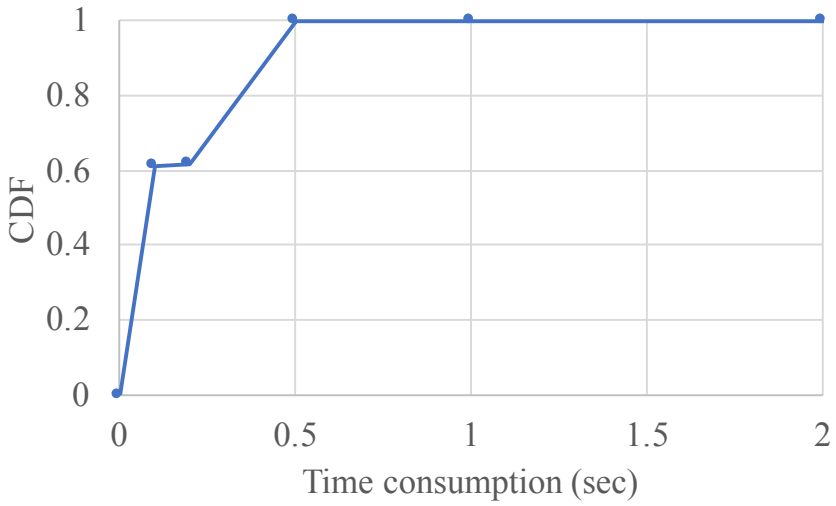

図6.4: 処理時間の累積分布 (タイムアウトした場合を含まない)

x86 命令ではどのシステムコールを発行するか eax レジスタの値によって決まる．シェルコードから痕跡を抽出できなった際はこの eax レジスタの値が存在しないシステムコールの値になっていたため，痕跡を観測できなった．

誤判定率は 0.01% 程度であった．誤判定した原因は通常の通信で偶然にもシェルコードの特徴である NOP スレッドや GetPC コードと同じバイト列が発生したためであった．

6.4.2 性能評価

D_{edb} および D_{msf} のシェルコードに対してエミュレーションに要した時間を計測した結果を図 6.4に示す．約 18% のシェルコードのエミュレーションはタイムアウト (5 秒) を引き起こしていたため，図 6.4はタイムアウトを引き起こさない場合の処理時間である．シェルコードの全体約 82% のシェルコードは 1 秒以内に判定が終了することから実環境でも多くの攻撃を瞬時に処理することが可能な性能である．タイムアウトの原因は接続を待ち受ける動作をす

表6.5: MACCDC データセットに適用した結果

年	2010	2011	2012
TCP セッション数	4,862,720	2,575,472	715,430
攻撃 TCP セッション数	11,215	16,233	25,640
攻撃成功判定数	2	0	0
攻撃失敗判定数	4,204	5	137
アラート削減割合	37.5%	0.03%	0.53%

るシェルコードによって接続待ちになってしまうことおよび，6.4.1節で述べたでエミュレーション環境で正しい番号のシステムコールの発行が行われず，ループ処理に陥ってしまうからである．

6.4.3 事例分析

セキュリティコンテストである MACCDC (Mid-Atlantic Collegiate Cyber Defense Competition) のネットワーク通信に対し，提案手法を適用した際の結果を示す．MACCDC はアメリカで開催されるコンテストであり，コンテスト運営者が攻撃者となり，大学生等から構成される参加チームのサーバに対して攻撃を行い，攻撃に対して参加チームがいかに迅速に対処するかを競うコンテストである．ただし，このデータセットには指標ラベル等が付与されておらず，ネットワーク環境を明記した資料もないため，精度等の評価にはできず，適用した際の結果のみを示す．2010 年から 2012 年までのコンテストの通信は PCAP ファイルとして公開されている [107] ためこれらの通信に対して，提案手法を適用し，判定結果を表 6.5に示す．攻撃 TCP セッション数はシェルコードを含む攻撃の数を表している．アラート削減割合は仮に全ての攻撃 TCP セッションを IDS が検知した際に，提案手法がアラートに対して失敗判定を下した割合である．2010 年において 2 件攻撃が成功したと判定した．いずれも指定された宛先 IP アドレス 157.99.66.56 のポート番号 5555 への接続を行う ELF[*1] ファイルを Web サーバからダウンロードした後，接続を行なっていた．分析の結果この 2 件の事象はユーザが故意にプログラムをダウンロードし実行したの可能性もあり，どちらも明確に攻撃を示唆するものではなかった．

2010 年および 2012 年に関しては攻撃を失敗として判定できていた．特に 2010 年においてはシェルコードを利用する攻撃全体に対して 36.8% も提案手法で自動的に攻撃が失敗と判定したことから実環境においてもアラートの削減に一定の効果をもたらす可能性を示した．

[*1] Linux の実行ファイルの形式

表6.6: MACCDC データセットから抽出した攻撃の痕跡

年	動作	IP アドレス	ポート番号
2010	connect	217.22.112.14	4721, 8925, 12517, 34112
	connect	10.0.2.15	2244
	connect	157.99.66.59	4454
	connect	217.22.112.53	29797
	connect	157.99.66.56	5555
	bind	150.204.83.15	4186, 22990, 28369
2011	connect	192.168.202.172	1465, 1543, 3035, 3171, 3559
2012	connect	192.168.202.102	43248, 47191, 48351, 49124, 49305, 50049, 50413, 53022

提案手法が抽出した痕跡を表 6.6に示す．痕跡を抽出できた攻撃のうち，2010 年の攻撃は IMAP サーバに対する攻撃，2012 年の攻撃は FTP サーバを対象とした攻撃であり，送信元 IP アドレスが 192.168.202.102 で，送信先 IP アドレスが 192.168.24.101 であった．2012 年のコンテストの開催資料 [108] から，192.168.21.0/24 から 192.168.28.0/24 の各/24 セグメントは 8 つの参加チームに与えられるサーバのネットワークであるため，参加チームが攻撃者からの攻撃を受けていたことが推測できる．提案手法はこれらの攻撃による侵害はないと自動的に判定することを可能にした．

6.5 制約

提案手法は攻撃に利用されるシェルコードを抽出し，そのシェルコードをエミュレーションすることで攻撃が成功した際の痕跡を実際の通信と比較して成否を判定するものである．その性質上以下のような場合には対応できない．

攻撃の痕跡が通信に現れない クライアントからのリクエストに対するレスポンスに攻撃の痕跡が残る場合は，既存手法で対処可能であるため本稿では対象としないが，サーバ内をファイル等を改変する攻撃においてはエミュレーションすることは可能だが，通信の観測ではサーバ内の状況の変化を把握できないため成否判定ができない．6.4節で用いたデータセット D_{edb} から推測すると，攻撃の痕跡が通信に現れないシェルコードは 783 (898-115) 件あり，シェルコード全体の 87.1% も存在する．

シェルコードが存在しない バックドアがソースコードに仕込まれていて，特殊なリクエストを送信することでバックドアを起動し接続を行う場合，バックドアの通信を観測するこ

とはできるが，シェルコードを観測することができないため，エミュレーションができず痕跡を把握できないため，成否判定を行うことはできない．また，権限昇格・認証バイパス等の認可・認証に関わる攻撃は通信から状況の変化を区別できないため対応できない．

ROP 形式のシェルコード　バッファオーバーフロー等によるシェルコードを防ぐため，OS にはデータ実行防止機構 (DEP: Data Execution Prevention) が備わっている [109]．DEP を回避し，シェルコードを実行することを目的とした攻撃手法が ROP (Return-Oriented Programming) である [110]．ROP の場合攻撃リクエストにはメモリアドレスが現れるため，6.3.1節で述べた抽出手法ではシェルコードと認識できず，成否判定できない．

6.5.1　関連研究

攻撃があった際に，その攻撃コードの挙動をエミュレートし，実行する機械語命令列を抽出する手法である [71, 72, 96]．ホスト内でも同じ機械語命令列の実行を観測した場合，攻撃が実行されたとしてアラートを出力することで攻撃が成功したときのみを判定することが可能となる．これらの手法は x86 アセンブリの攻撃コードのエミュレーションに限られるが，Web アプリケーションに対しては，コードインジェクションの攻撃に特化した WeXpose [73] が存在する．ただ，この手法ではエミュレーション後の挙動を実システム内で観測する必要があるため，導入先の改変が必要である．また，著者がすでに提案した手法 [69] はシステム改変は必要ないが，攻撃リクエストに対するレスポンスに痕跡がある場合にしか対応できない．しかしこの手法では攻撃リクエストに対するレスポンスに痕跡がある場合のみに成否判定が可能となる．本研究はレスポンス以外の通信に攻撃の痕跡がある場合に対応しており，既存研究と異なる．

6.6　結語

大量にあるアラートに対して効率的に処理するという課題に対して，本研究では攻撃の成否判定を行うことによってアラートの優先度を分別することで課題の解決に貢献した．提案手法では攻撃に利用されるシェルコードをエミュレーションし，エミュレーションの結果得られる攻撃痕跡と実際の通信の挙動を照合することで攻撃の成否を判定した．評価よりリモート型シェルコードの約 60.0% を正しく判定可能であった．今後はエミュレータの多様化やシェルコードの抽出精度の向上により判定カバレッジを向上していく．

第 7 章

結論

7.1 まとめ

本研究では，Web アプリケーションに対する攻撃のより効率的な検知を実現するために，IDS の攻撃検知アラートに攻撃の成否に関する情報が必要であるおよび攻撃の原因と被害に関する情報が必要であるという 2 つの要件を掲げ，これらの要件を満たすために取り組む必要がある 3 つの課題を挙げた．それぞれ，NIDS における誤検知アラートが発生する課題，HIDS におけるアラートに攻撃の情報が関連づかない課題，NIDS におけるアラートに攻撃の成否に関する情報がない課題の 3 つである．これらの課題を鑑み，Web アプリケーションに対する攻撃をより効率的に検知する仕組みについて検討した．

NIDS の誤検知の課題については異常検知に基づくより正確な攻撃検知アルゴリズムの提案し，HTTP リクエストの各要素の文字列の構造に着目し，学習を行うことで，誤検知を低減した．評価の結果，既存の異常検知手法に比べ提案手法の方が精度が高く，また性能も実用的な処理速度であることを示した．

HIDS におけるアラートに攻撃の原因である HTTP リクエストと関連づかないという課題については HIDS アラートと HTTP リクエストの関連付け手法を提案した．HIDS の入力であるシステムコールや SQL クエリ発行などのイベントとそれらを発生させた HTTP リクエストと関連付けを行うことで，HIDS で検知した際に管理者が攻撃対象の Web アプリケーションの URL や攻撃元の IP アドレスを特定できるようにした．評価では，提案手法が誤った関連付けをすることがなく，Web アプリケーションに与えるパフォーマンス低下を 5% 程度に抑えた実用的な手法であることを示した．

NIDS におけるアラートに攻撃の成否に関する情報がないという課題については攻撃コードのエミュレーションを行い，攻撃成功時の痕跡を抽出し，この痕跡が HTTP レスポンスに含まれるか否かで攻撃の成否を判定する手法を提案した．提案手法の評価結果，および発見した攻撃事例から提案手法の実用性を示した．

その結果，Web アプリケーションに対する攻撃においてサーバ管理者や SOC 分析者の対応が不要となる攻撃によるアラート通知を減少させたり，アラートの対処に必要な情報を関連づけることに貢献した．今後も Web アプリケーションに対するサイバー攻撃が増加し続けることが予想されるが，本研究で提案する手法を IDS に取り入れることで，管理者や SOC 分析者の対応コストの低減につながることが期待できる．

7.2　今後の展望

本論文では，Web アプリケーションに対する攻撃の検知について論じ，その際の課題として，正常なアクセスを攻撃として検知してしまう誤検知や攻撃として検知するだけでなく，攻撃の成否についても判定する必要があることについて述べた．Web アプリケーションの利用は今後も続き，Web アプリケーションに対する攻撃は今後ますます増加することが予想され，新しい手口の攻撃も発生すると考えられる．また，セキュリティ分野における固有の問題に，対策手法を公開することでその対策を回避する攻撃手法が発生することもある．攻撃の変化以外にも Web アプリケーション自体の変化も発生する．例えば，Web アプリケーションの通信で利用される HTTP もより効率的な形態である HTTP/2 への移行が進行すると予想される．

攻撃の手口の変化に関しては，3章にて提案した手法は Web アプリケーションの正常アクセスを学習するため，攻撃の傾向に依存しないことが予想されるが，5章および 6章にて提案した Web アプリケーションの成否を判定する手法では攻撃の意図に応じてその成否を判定するアルゴリズムが必要になってしまう．そのため，新しい手口の攻撃が発見されると，手法の改良が必要になる可能性がある．

提案した手法を回避するような攻撃手法の発生は実装面によっては可能である考える．例えば，4章で提案した手法ではログ量の多さが課題となるが，攻撃者がこの性質を知っていれば大量にアクセスを行うことで大量のログによってディスクを溢れさせて，提案手法の機能を制限するが可能なため，今後も検討を続けていく必要がある．

HTTP/2 への移行により，通信路の強制的な暗号化，フレームとストリームによる通信の多重化，サーバプッシュの機能によりリクエストとレスポンスが非同期に発生することとなる．3章，5章および 6章で提案した手法はネットワーク通信を観測する必要があるため，観測点を暗号化を解除した後にする必要がある．また，4章で提案した手法は HTTP/2 の同時接続性向上により，関連付けの精度に影響する可能性があるため今後，検証を行っていく必要があると考える．しかし，通信をリクエストとレスポンスで行うという概念は変化しない．またヘッダ圧縮の機能によってリクエストとレスポンスのヘッダ部の送受信量が必要最小限となるが，情報量という観点では今までと変化しないため，3章，5章および 6章で提案した手法は HTTP/2 移行によって実装面での変更はあるものの，手法自体がすぐに陳腐化してしまうことはない考える．

セキュリティ分野は攻撃手法が進化すれば防御手法も進化し，防御手法が進化すれば攻撃手法も進化するといういたちごっこの状態である．しかし，企業や組織がコストをかけて行なったセキュリティ対策がいとも簡単にも破られてしまう事件があったり，Web アプリケーションをはじめとした IT 技術の進化に防御手法が追随する必要があるため，近年では，攻撃者優位という認識が浸透しつつある．今後は攻撃手法や新しい IT 技術に追随し，攻撃者優位を覆すような防御手法に関する研究の更なる加速が必要であると考える．例えば，本研究で対象としているのは効率的な検知の部分であるが，今後は，サイバー攻撃の検知から対処まで人手を介さないで自動的に処理される技術の研究をすることで，この社会問題の解決に貢献していきたい．

謝辞

本研究を行い本論文の執筆するにあたり，非常に多くの方に多大なるご支援ご指導をいただきました．心より御礼申し上げます．

本論文は，著者が京都大学大学院情報学研究科知能情報学専攻博士後期課程および日本電信電話株式会社セキュアプラットフォーム研究所に在籍中の研究成果をまとめたものです．本研究を進め，本論文を執筆するにあたりまして，指導教員の岡部寿男教授には多大なるご指導ご助言を賜りました．心より感謝申し上げます．また，論文審査委員 (副査) として貴重なご助言を賜りました森信介教授ならびに五十嵐淳教授に深く御礼申し上げます．

本論文の各研究テーマを進めるにあたりまして，宮崎修一准教授，小谷大祐助教，国立情報学研究所の高倉弘喜教授，名古屋大学情報基盤センターの嶋田創准教授，そして研究室の皆様には多大なるご助言を賜りました．深くお礼申し上げます．

職場で働きながら学位取得を目指せる環境を作ってくだり全面的にご協力くださいました日本電信電話株式会社セキュアプラットフォーム研究所元所長の大久保一彦氏，所長の平田真一氏，元プロジェクトマネージャの針生剛男氏，プロジェクトマネージャの早川和宏氏，グループリーダの三好潤氏，波戸邦夫氏に深謝いたします．研究の指針や技術課題に関して適切なご助言・ご指導を頂いた同研究所の青木一史氏，岩村誠特別研究員に深く感謝申し上げます．また，博士課程に進むきっかけを与えてくださった同研究所の秋山満昭上席特別研究員，並びにNTT セキュリティ・ジャパン株式会社の八木毅氏に深く感謝いたします．

最後に，学位取得に向けて常に応援し支えてくれた妻の有美に心より感謝します．

参考文献

[1] Christopher Kruegel and Giovanni Vigna. Anomaly Detection of Web-based Attacks. In *ACM CCS*, pp. 251–261, 2003.

[2] Christopher Kruegel, Giovanni Vigna, and William Robertson. A Multi-Model Approach to the Detection of Web-Based Attacks. *Computer Networks*, Vol. 48, No. 5, pp. 717–738, 2005.

[3] Tae Hyung Kim, Kyungtae Kim, Jong Kim, and Sung Je Hong. Profile-based Web Application Security System with Positive Model Selection. In *Proceedings of the 2nd Joint Workship on Information Security*, 2007.

[4] Ke Wang, Janak J Parekh, and Salvatore J Stolfo. Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In *RAID*, pp. 226–248, 2006.

[5] Kenneth L Ingham and Hajime Inoue. Comparing Anomaly Detection Techniques for HTTP. In *RAID*, pp. 42–62, 2007.

[6] Yuxuan Luo, Shaoyin Cheng, Chong Liu, and Fan Jiang. PU Learning in Payload-based Web Anomaly Detection. In *Security of Smart Cities, Industrial Control System and Communications (SSIC)*, pp. 1–5, 2018.

[7] Gustavo Betarte, Eduardo Giménez, Rodrigo Martínez, and Álvaro Pardo. Improving Web Application Firewalls through Anomaly Detection. In *IEEE ICMLA*, pp. 779–784, 2018.

[8] Timothy J Berners-Lee and Robert Cailliau. WorldWideWeb: Proposal for a HyperText project, 1990. `https://www.w3.org/Proposal`, (last visited 2020-1-7).

[9] Netcraft. December 2019 Web Server Survey, 2019. `https://news.netcraft.com/archives/2019/12/10/december-2019-web-server-survey.html`, (last visited 2020-1-7).

[10] WordPress.com: Create a Free Website or Blog. `https://ja.wordpress.com`, (last visited 2019-1-21).

[11] Joomla Content Management System. `https://www.joomla.org`, (last visited 2019-

1-21).

[12] Welcome! - The Apache HTTP Server Project. https://httpd.apache.org, (last visited 2019-1-21).

[13] NGINX: High Performance Load Balancer, Web Server and Reverse Proxy. https://www.nginx.com, (last visited 2019-1-21).

[14] The HTTP Protocol As Implemented In W3. https://www.w3.org/Protocols/HTTP/AsImplemented.html, (last visited 2019-1-21).

[15] HTTP/2 Approved. https://www.ietf.org/blog/http2-approved/, (last visited 2019-1-21).

[16] Common Vulnerabilities and Exposures. https://cve.mitre.org, (last visited 2020-1-7).

[17] Mike Samuel, Prateek Saxena, and Dawn Song. Context-Sensitive Auto-Sanitization in Web Templating Languages Using Type Qualifiers. In *ACM CCS*, pp. 587–600, 2011.

[18] Davide Balzarotti, Marco Cova, Viktoria Felmetsger, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. *IEEE S&P*, pp. 387–401, 2008.

[19] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Static Analysis for Detecting Taint-Style Vulnerabilities in Web Applications. *Computer Security*, Vol. 18, No. 5, pp. 861–907, 2010.

[20] Johannes Dahse and Thorsten Holz. Simulation of Built-in PHP Features for Precise Static Code Analysis. In *NDSS*, 2014.

[21] Davide Canali and Davide Balzarotti. Behind the Scenes of Online Attacks: an Analysis of Exploitation Behaviors on the Web. In *NDSS*, 2013.

[22] John P John, Fang Yu, Yinglian Xie, Martin Abadi, and Arvind Krishnamurthy. Searching the Searchers with Searchaudit. *USENIX Security*, p. 9, 2010.

[23] Positive Technologies. WEB APPLICATION ATTACK STATISTICS. https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Web-application-attacks-2018-eng.pdf, (last visited 2020-1-7).

[24] Akamai. State of The Internet Security Report. https://www.akamai.com/us/en/resources/our-thinking/state-of-the-internet-report/, (last visited 2020-1-7).

[25] Yi Xie and Shun-zheng Yu. Monitoring the Application-Layer DDoS Attacks for Popular Websites. *IEEE/ACM Transactions on Networking (TON)*, Vol. 17, No. 1,

pp. 15–25, 2009.

[26] Enrico Cambiaso, Gianluca Papaleo, and Maurizio Aiello. Taxonomy of Slow DoS Attacks to Web Applications. In *SNDS*, pp. 195–204, 2012.

[27] OWASP Top 10 - 2017. `https://www.owasp.org/images/2/23/OWASP_Top_10-2017(ja).pdf`, (last visited 2019-2-6).

[28] Ke Wang and Salvatore J Stolfo. Anomalous Payload-based Network Intrusion Detection. *RAID*, pp. 203–222, 2004.

[29] Jungsuk Song, Kenji Ohira, Hiroki Takakura, Yasuo Okabe, and Yongjin Kwon. A Clustering Method for Improving Performance of Anomaly-Based Intrusion Detection System. *IEICE Transactions on Information and Systems*, Vol. E91-D, No. 5, pp. 1282–1291, 2008.

[30] Jungsuk SONG, Hiroki TAKAKURA, Yasuo OKABE, and Yongjin KWON. Unsupervised Anomaly Detection Based on Clustering and Multiple One-Class SVM. *IEICE Transactions on Communications*, Vol. 92, No. 6, pp. 1981–1990, 2009.

[31] Yingbo Song, Angelos D Keromytis, and Salvatore J Stolfo. Spectrogram : A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic. In *NDSS*, 2009.

[32] Kenneth L Ingham, Anil Somayaji, John Burge, and Stephanie Forrest. Learning DFA Representations of HTTP for Protecting Web Applications. *Computer Networks*, Vol. 51, No. 5, pp. 1239–1255, 2007.

[33] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion Detection using Sequences of System Calls. *Computer Security*, Vol. 6, No. 3, pp. 151–180, 1998.

[34] David Wagner and Paolo Soto. Mimicry Attacks on Host-Based Intrusion Detection Systems. *ACM CCS*, pp. 255–264, 2002.

[35] Darren Mutz, Fredrik Valeur, Giovanni Vigna, and Christopher Kruegel. Anomalous System Call Detection. *ACM Transactions on Information and System Security (TISSEC)*, Vol. 9, No. 1, pp. 61–93, 2006.

[36] William G J Halfond and Alessandro Orso. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In *IEEE/ACM ASE*, pp. 174–183, 2005.

[37] Fredrik Valeur, Darren Mutz, and Giovanni Vigna. A Learning-Based Approach to the Detection of SQL Attacks. In *DIMVA*, pp. 123–140, 2005.

[38] National Institute of Standards and Technology. Framework for Improving Critical Infrastructure Cybersecurity. `https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf`, (last visited 2019-1-28).

[39] Cisco. 年次サイバーセキュリティレポート, 2017. `https://www.cisco.com/c/dam/`

m/ja_jp/offers/173/sc04/cisco-annual-cybersecurity-report-2017.pdf, (last visited 2019-12-26).

[40] Tianqi Chen and Carlos Guestrin. XGBoost : A Scalable Tree Boosting System. In *KDD*, pp. 785–794, 2016.

[41] Federico Maggi, William Robertson, Christopher Kruegel, and Giovanni Vigna. Protecting a Moving Target : Addressing Web Application Concept Drift. In *RAID*, pp. 21–40, 2009.

[42] Lasse Bergroth, Harri Hakonen, and Timo Raita. A Survey of Longest Common Subsequence Algorithms. In *IEEE SPIRE*, p. 39, 2000.

[43] Ron Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *IJCAI*, pp. 1137–1143, 1995.

[44] Ryan Barnett. WAF Virtual Patching Challenge: Securing WebGoat with ModSecurity. *Breach Security*, 2009.

[45] Martin Roesch. Snort: Lightweight Intrusion Detection for Networks. In *LISA*, pp. 229–238, 1999.

[46] Emergence Threats Rule. `https://rules.emergingthreats.net`, (last visited 2020-1-7).

[47] PyPy. `https://pypy.org`, (last visited 2020-1-7).

[48] James A Hanley and Barbara J McNeil. The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve. In *Radiology*, Vol. 30, pp. 1145–1159, 1982.

[49] pingdom. `https://www.pingdom.com/`, (last visited 2020-1-7).

[50] Michiaki Ito and Hitoshi Iyatomi. Web Application Firewall using Character-level Convolutional Neural Network. In *IEEE CSPA*, pp. 103–106, 2018.

[51] Jingxi Liang, Wen Zhao, and Wei Ye. Anomaly-Based Web Attack Detection : A Deep Learning Approach. In *ACM ICNCC*, pp. 80–85, 2017.

[52] Hieu Mac, Bach Khoa, and Cybersecurity Centre. Detecting Attacks on Web Applications using Autoencoder. In *ACM SoICT*, pp. 416–421, 2018.

[53] William K Robertson, Federico Maggi, Christopher Kruegel, and Giovanni Vigna. Effective Anomaly Detection with Scarce Training Data. In *NDSS*, 2010.

[54] Federico Maggi, Student Member, Matteo Matteucci, and Stefano Zanero. Detecting Intrusions through System Call Sequence and Argument Analysis. *IEEE Transactions on Dependable and Secure Computing*, Vol. 7, No. 4, pp. 381–395, 2010.

[55] Robert A. Bridges, Tarrah R. Glass-Vanderlan, Michael D. Iannacone, and Maria S. Vincent. A Survey of Intrusion Detection Systems Leveraging Host Data. *arXiv*,

2019.

[56] Florian Skopik and Roman Fiedler. Intrusion Detection in Distributed Systems using Fingerprinting and Massive Event Correlation. In *GI-Jahrestagung*, pp. 2240–2254, 2013.

[57] Samuel T King and Peter M Chen. Backtracking Intrusions. *ACM SOSP*, Vol. 37, No. 5, pp. 223–236, 2005.

[58] D. A. Menasce. Web Server Software Architectures. *IEEE Internet Computing*, Vol. 7, No. 6, pp. 78–81, 2003.

[59] SystemTap. `https://sourceware.org/systemtap`, (last visited 2020-1-7).

[60] Vara Prasad, William Cohen, F C Eigler, Martin Hunt, Jim Keniston, and J Chen. Locating System Problems Using Dynamic Instrumentation. In *Ottawa Linux Symposium*, 2005.

[61] Frank Ch Eigler. Problem Solving With Systemtap. In *Ottawa Linux Symposium*, 2006.

[62] Bart Jacob, Paul Larson, B Leitao, and SAMM Da Silva. SystemTap: Instrumenting the Linux Kernel for Analyzing Performance and Functional Problems. *IBM Redbook*, 2008.

[63] Apache JMeter. `https://jmeter.apache.org/`, (last visited 2020-1-7).

[64] Nielsen Norman Group. Website Response Times. `https://www.nngroup.com/articles/website-response-times/`, (last visited 2020-1-7).

[65] Jim Keniston, Ananth Mavinakayanahalli, Prasanna Panchamukhi, and Vara Prasad. Ptrace, Utrace, Uprobes: Lightweight, Dynamic Tracing of User Apps. In *Linux Symposium*, 2007.

[66] 企業における情報システムのログ管理に関する実態調査. 独立行政法人情報処理推進機構 (IPA), 2016.

[67] 高度サイバー攻撃への対処におけるログの活用と分析方法. 一般社団法人 JPCERT コーディネーションセンター, 2016.

[68] Tripwire Enterprise. `https://www.tripwire.co.jp/products/enterprise/`, (last visited 2020-1-7).

[69] 鐘本楊, 青木一史, 三好潤, 嶋田創, 高倉弘喜. 攻撃コードのエミュレーションに基づく Web アプリケーションに対する攻撃の成否判定手法. 情報処理学会論文誌, Vol. 60, No. 3, pp. 945–955, 2019.

[70] 鐘揚, 佐藤徹, 谷川真樹. AETP: イベントの関連づけによる Web アプリケーションに対する有効な攻撃の検知手法. コンピュータセキュリティシンポジウム 2016 論文集, pp. 943–950, 2016.

[71] Ali Abbasi, Jos Wetzels, Wouter Bokslag, Emmanuele Zambon, and Sandro Etalle. On Emulation-Based Network Intrusion Detection Systems. In *RAID*, pp. 384–404, 2014.

[72] Michalis Polychronakis, Kostas G Anagnostakis, and Evangelos P Markatos. Network-Level Polymorphic Shellcode Detection Using Emulation. In *DIMVA*, pp. 257–274, 2006.

[73] Jennifer Bellizzi and Mark Vella. WeXpose : Towards on-Line Dynamic Analysis of Web Attack Payloads using Just-In-Time Binary Modification. In *SECRYPT*, pp. 5–15, 2015.

[74] Giovanni Vigna, William Robertson, Vishal Kher, and Richard A Kemmerer. A Stateful Intrusion Detection System for World-Wide Web Servers. In *ACSAC*, p. 34, 2003.

[75] Robin Sommer and Vern Paxson. Enhancing Byte-level Network Intrusion Detection Signatures with Context. In *ACM CCS*, pp. 262–271, 2003.

[76] Jingmin Zhou, Adam J Carlson, and Matt Bishop. Verify Results of Network Intrusion Alerts Using Lightweight Protocol Analysis. In *ACSAC*, pp. 117–126, 2005.

[77] Martin Johns, B Engelmann, and Joachim Posegga. XSSDS: Server-Side Detection of Cross-Site Scripting Attacks. In *ACSAC*, pp. 335–344, 2008.

[78] QEMU, the FAST! processor emulator. `http://www.qemu.org/`, (last visited 2020-1-7).

[79] Docker, Debug your app, not your environment. `https://www.docker.com/`, (last visited 2020-1-7).

[80] The Zeek Network Security Monitor. `https://www.zeek.org`, (last visited 2020-1-7).

[81] fuzzdb project. FuzzDB. `https://github.com/fuzzdb-project/fuzzdb`, (last visited 2020-1-7).

[82] foospidy. Git All the Payloads! A collection of web attack payloads. `https://github.com/foospidy/payloads`, (last visited 2020-1-7).

[83] Alexa. The top 500 sites on the web. `https://www.alexa.com/topsites`, (last visited 2020-1-7).

[84] Neminath Hubballi and Vinoth Suryanarayanan. False Alarm Minimization Techniques in Signature-based Intrusion Detection Systems: A Survey. *Computer Communications*, Vol. 49, pp. 1–17, 2014.

[85] Frédéric Cuppens and Alexandre Miège. Alert Correlation in a Cooperative Intrusion Detection Framework. In *IEEE S&P*, pp. 202–215, 2002.

[86] Helen J Wang, Chuanxiong Guo, Daniel R Simon, and Alf Zugenmaier. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In *SIGCOMM*, pp. 193–204, 2004.

[87] David Brumley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. Towards Automatic Generation of Vulnerability-Based Signatures. In *IEEE S&P*, pp. 15–16, 2006.

[88] Frederic Massicotte, Francois Gagnon, Yvan Labiche, Lionel Briand, and Mathieu Couture. Automatic Evaluation of Intrusion Detection Systems. *ACSAC*, pp. 361–370, 2006.

[89] Frederic Massicotte, Yvan Labiche, and Lionel C Briand. Toward Automatic Generation of Intrusion Detection Verification Rules. In *ACSAC*, pp. 279–288, 2008.

[90] Christopher Kruegel and William Robertson. Alert Verification: Determining the Success of Intrusion Attempts. In *DIMVA*, 2004.

[91] Frederic Massicotte and Mathieu Couture. Context-Based Intrusion Detection Using Snort, Nessus and Bugtraq Databases. In *PST*, 2005.

[92] Fredrik Valeur, Giovanni Vign, Christopher Kruegel, and Richard A Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. In *IEEE DSC*, Vol. 1, pp. 146–169, 2004.

[93] David Swift. A Practical Application of SIM/SEM/SIEM Automating Threat Identification. `https://www.sans.org/reading-room/whitepapers/logging/practical-application-sim-sem-siem-automating-threat-identification-1781`, (last visited 2019-1-28).

[94] IBM QRadar SIEM. `https://www.ibm.com/jp-ja/marketplace/ibm-qradar-siem`, (last visited 2019-1-28).

[95] The Data-to-Everything Platform. `https://www.splunk.com/en_us`, (last visited 2019-1-28).

[96] 嶋村誠, 河野健二. Yataglass: 攻撃の擬似実行による攻撃メッセージの振舞いの解析. 情報処理学会論文誌, Vol. 50, No. 9, pp. 2371–2381, 2009.

[97] Ramkumar Chinchani and Eric Van Den Berg. A Fast Static Analysis Approach To Detect Exploit Code Inside Network Flows. In *RAID*, pp. 284–308, 2005.

[98] J.C. Foster. Sockets, Shellcode, Porting, and Coding: Reverse Engineering Exploits and Tool Coding for Security Professionals. *Elsevier Science*, 2005.

[99] 藤井孝好, 吉岡克成, 四方順司, 松本勉. エミュレーションに基づくシェルコード検知手法の改善. マルウェア対策研究人材育成ワークショップ (MWS), 2010.

[100] 中尾康二, 神保千晶, 吉岡克成, 四方順司, 松本勉, 衛藤将史, 井上大介. CPU エミュレ

ータと Dynamic Binary Instrumentation の併用によるシェルコード動的分析手法の提案. 電子情報通信学会技術研究報告 (ICSS), pp. 59–64, 2010.

[101] 田中恭之, 後藤厚宏. 攻撃コードの特徴からみた対策の検討. 暗号と情報セキュリティシンポジウム (SCIS), 2014.

[102] Nguyen Anh Quynh and Dang Hoang Vu. Unicorn: Next Generation CPU Emulator Framework Self-introduction. In *BlackHat USA*, 2015.

[103] buffer. libemu: x86 emulation and shellcode detection. `https://github.com/buffer/libemu`, (last visited 2020-1-7).

[104] Offensive Security. Exploit Database. `https://exploit-db.com/`, (last visited 2018-7-25).

[105] Rapid7. metasploit. `https://www.metasploit.com/`, (last visited 2019-1-7).

[106] Wireshark SampleCaptures. `https://wiki.wireshark.org/SampleCaptures/`, (last visited 2018-8-1).

[107] Capture files from Mid-Atlantic CCDC. `https://www.netresec.com/?page=MACCDC`, (last visited 2019-1-7).

[108] National CyberWatch Center. Preparing for the Collegiate Cyber Defense Competition (CCDC). `https://www.nationalcyberwatch.org/ncw-content/uploads/2016/03/NCC_Press_How_To_Prepare_For_the_CCDC-1.pdf`, (last visited 2019-1-7).

[109] Microsoft. Data Execution Prevention. `https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention`, (last visited 2019-1-28).

[110] Marco Prandini and Marco Ramilli. Return-Oriented Programming. In *IEEE S&P*, pp. 84–87, 2012.

発表論文一覧

学位論文対象の論文誌

1. **鐘本楊**，青木一史，三好潤，嶋田創，高倉弘喜，攻撃コードのエミュレーションに基づく Web アプリケーションに対する攻撃の成否判定手法，情報処理学会論文誌，No.60，Vol.3，pp.945–955，2019
2. **鐘本楊**，青木一史，三好潤，嶋田創，高倉弘喜，文字列構造に着目した Web アプリケーションに対する攻撃のアノマリ検知手法，情報処理学会論文誌，No.60，Vol.12，pp. 2223–2233，2019
3. **鐘本楊**，青木一史，三好潤，小谷大祐，岡部寿男，HIDS アラート調査のための HTTP リクエストとホストイベントの関連付け手法，情報処理学会論文誌，No.61，Vol.5 (掲載予定)，2020

学位論文対象の国際会議

1. **Yo Kanemoto**, Kazufumi Aoki, Makoto Iwamura, Jun Miyoshi, Daisuke Kotani, Hiroki Takakura, Yasuo Okabe, Detecting Successful Attacks from IDS Alerts Based on Emulation of Remote Shellcodes, Proc. 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Vol.2, pp.471-476, July 2019.

その他の論文誌

1. 千田忠賢，**鐘本楊**，青木一史，三好潤，行動表現文法 ELL を用いた攻撃シナリオ再構築技術，情報処理学会論文誌，No.61，Vol.2（掲載予定），2020

その他の国際会議

1. **Yang Zhong**, Hiroshi Asakura, Hiroki Takakura, Yoshihito Oshima, Detecting Malicious Inputs of Web Application Parameters Using Character Class Sequences, Proc. 2015 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC), Vol.2, pp.525-532, July 2015.

その他の国内研究会

1. **鐘揚**, 朝倉浩志, 高倉弘喜, 大嶋嘉人, Web アプリケーションのパラメタを悪用する攻撃のアノマリ検知手法, コンピュータセキュリティシンポジウム 2014 論文集, pp.474-481, 2014
2. **鐘揚**，朝倉浩志，谷川真樹，大嶋嘉人，攻撃とその影響を特定可能とする Web サーバログ相関手法，コンピュータセキュリティシンポジウム 2015 論文集，pp.132-139，2015
3. **鐘揚**，佐藤徹，谷川真樹，AETP: イベントの関連づけによる Web アプリケーションに対する有効な攻撃の検知手法，コンピュータセキュリティシンポジウム 2016 論文集，pp.943-950，2016
4. **鐘揚**，折原慎吾，谷川真樹，嶋田創，村瀬勉，高倉弘喜，大嶋嘉人，URI の共起性検知に基づく Web スキャンの実態調査，電子情報通信学会技術研究報告 (ICSS) ，No.488, Vol.115，pp.25-30，2016
5. **鐘揚**，佐藤徹，谷川真樹，Web アプリケーションの静的ソースコード解析によるホスト型攻撃検知の誤検知低減に向けて，研究報告セキュリティ心理学とトラスト（SPT），pp.1-6，2017
6. **鐘揚**，青木一史，三好潤，嶋田創，高倉弘喜，AVT Lite: 攻撃コードのエミュレーションに基づく Web 攻撃の成否判定手法，コンピュータセキュリティシンポジウム 2017 論文集，2017
7. **鐘本楊**，青木一史，岩村誠，三好潤，小谷大祐，高倉弘喜，岡部寿男，リモート型シェルコードのエミュレーションによる攻撃成否判定手法，コンピュータセキュリティシンポジウム 2018 論文集，pp.425-432，2018
8. 千田忠賢，**鐘本楊**，青木一史，三好潤，行動表現文法 ELL を用いた攻撃シナリオ再構築技術，コンピュータセキュリティシンポジウム 2018 論文集，pp.433-440，2018
9. 黒木琴海，**鐘本楊**，青木一史，三好潤，アクセスパターンに基づく攻撃対象 Web アプリケーション発見手法の提案，コンピュータセキュリティシンポジウム 2018 論文集，

pp.71-76，2018

10. 守屋広汰，岡田隆三，西垣正勝，野口靖浩，黒木琴海，**鐘本楊**，猿渡俊介，HTTP ログを用いた Botnet からの Web スキャン検出手法の初期的検討，電子情報通信学会総合大会，2019
11. 黒木琴海，**鐘本楊**，青木一史，三好潤，野口靖浩，西垣正勝，構文解析とエミュレーションを組み合わせた SQL インジェクション被害推定手法，暗号と情報セキュリティシンポジウム（SCIS)，2020
12. 竹村太一，千田忠賢，**鐘本楊**，三好潤，中沢実，TTPs を活用した攻撃シナリオ関連付け手法，暗号と情報セキュリティシンポジウム（SCIS)，2020

その他の書籍

1. 折原慎吾，鐘本楊，神谷和憲，松橋亜希子，阿部慎司，永井信弘，羽田大樹，朝倉浩志，田辺英昭，セキュリティのためのログ分析入門―サイバー攻撃の痕跡を見つける技術―，技術評論社，2018

受賞

1. 2017 年度　コンピュータセキュリティシンポジウム 2017 奨励賞受賞，「AVT Lite: 攻撃コードのエミュレーションに基づく Web 攻撃の成否判定手法」，2017 年 10 月
2. 2018 年度　コンピュータセキュリティシンポジウム 2018 奨励賞受賞，「行動表現文法 ELL を用いた攻撃シナリオ再構築技術」，2018 年 10 月
3. 2018 年度　情報処理学会論文誌ジャーナル特選論文受賞，「攻撃コードのエミュレーションに基づく Web アプリケーションに対する攻撃の成否判定手法」，2019 年 3 月

©2015 IEEE. Reprinted, with permission, from Yang Zhong, Hiroshi Asakura, Hiroki Takakura, Yoshihito Oshima, Detecting Malicious Inputs of Web Application Parameters Using Character Class Sequences, Proc. 2015 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC), Vol.2, pp.525-532, July 2015. DOI:10.1109/COMPSAC.2015.73

©2019 IEEE. Reprinted, with permission, from Yo Kanemoto, Kazufumi Aoki, Makoto Iwamura, Jun Miyoshi, Daisuke Kotani, Hiroki Takakura, Yasuo Okabe, Detecting Successful Attacks from IDS Alerts Based on Emulation of Remote Shellcodes, Proc. 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Vol.2, pp.471-476, July 2019. DOI:10.1109/COMPSAC.2019.10251

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Kyoto University's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink.